



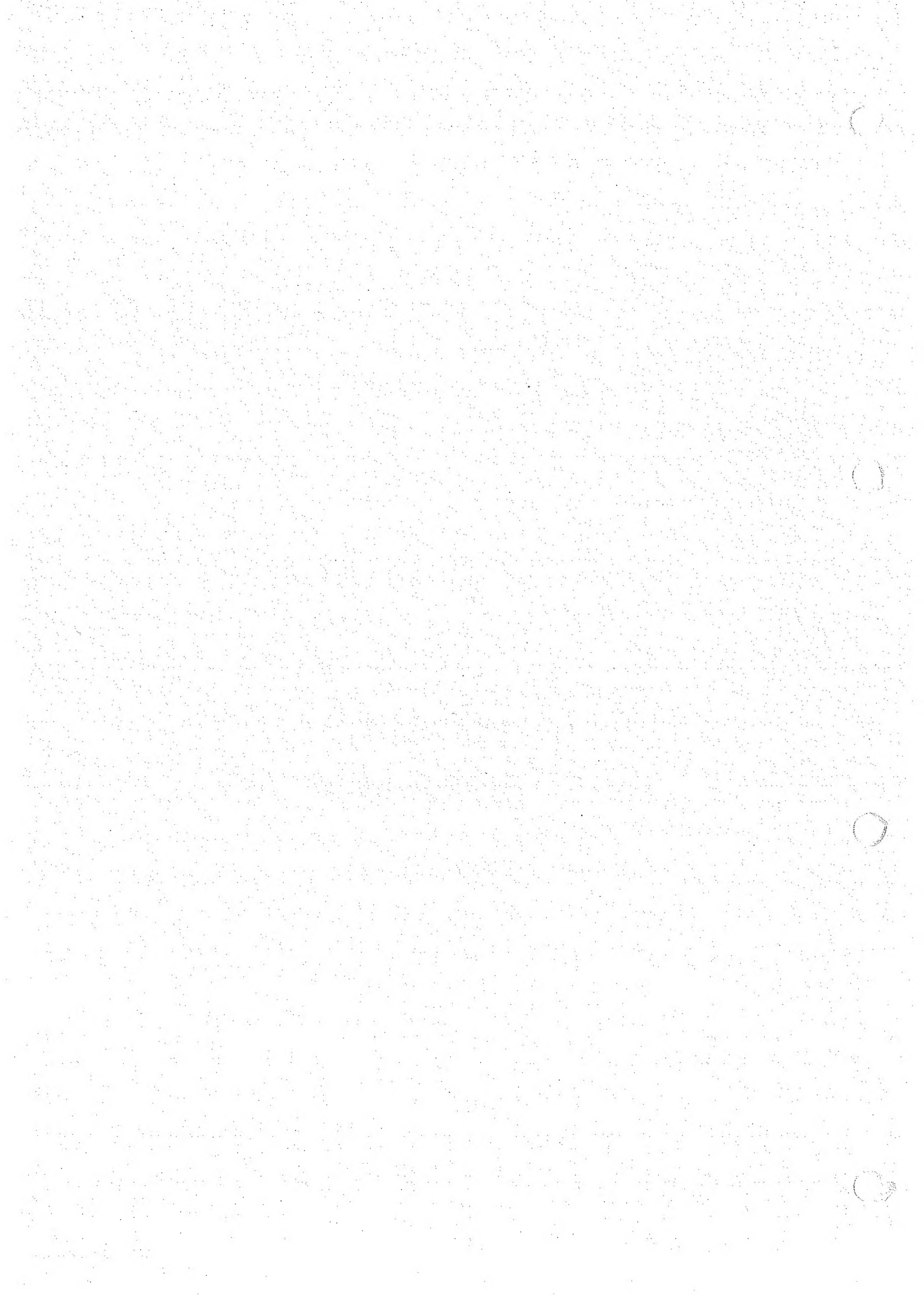
ULISSE s.r.l.
Servizio Distribuzione Apple
Via Pacini, 22
20131 Milano
Tel. (02) 2367783-2367406

su licenza esclusiva
Apple Computer S.p.A.

MPW C Language

Version 1.0

MPW20



Macintosh Programmer's Workshop C Reference

October 3, 1986

Apple Technical Publications

This document contains preliminary information. It does not include

- *final editorial corrections*
- *final art work*
- *an index.*

It may not include final technical changes.

• Apple Computer, Inc.

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or part, without written consent of Apple, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

You may use the software on any computer owned by you, but extra copies may not be made for this purpose.

© Apple Computer, Inc., 1986
20525 Mariani Ave.
Cupertino, California 95014
(408) 996-1010

Apple, the Apple logo, and LaserWriter are registered trademarks of Apple Computer, Inc.

Macintosh and SANE are trademarks of Apple Computer, Inc.

UNIX is a registered trademark of AT&T.

DEC, VAX, and PDP are trademarks of Digital Equipment Corporation.

IBM is a registered trademark of International Business Machines Company.

NS16000 is a trademark of National Semiconductor Corporation.

Z8000 and Z8070 are trademarks of Zilog Corporation.

Simultaneously published in the United States and Canada.



Contents



| | |
|------|--|
| vii | Figures and Tables |
| 1 | Preface |
| 1 | What this manual contains |
| 2 | Other reference material you'll need |
| 3 | Typographic and spelling conventions |
| 3 | Language notation |
| 3 | Boldface |
| 3 | Spelling and capitalization |
| | |
| 1 | Chapter 1: About MPW C |
| 2 | About the C Compiler |
| 3 | About the C libraries |
| 3 | Libraries shared with other languages: directory {Libraries} |
| 4 | C Libraries: directory {CLibraries} |
| 4 | About the C examples |
| 5 | Installing MPW C |
| 7 | Creating an application in C |
| 7 | Compiling an application |
| 8 | Include files |
| 9 | Segmentation control |
| 10 | Creating Resources |
| 10 | Linking an application |
| 11 | Creating MPW a tool in C |
| 12 | Compiling a tool |
| 12 | Linking a tool |
| 12 | The Integrated Environment |
| 13 | Creating a desk accessory in C |
| 13 | Desk accessory restrictions |
| 14 | The DRVRRuntime Library |
| 14 | Desk accessory routines |
| 15 | Linking a desk accessory |

| | |
|----|---|
| 1 | Chapter 2: The MPW C Language |
| 2 | Language definition |
| 2 | Variable names |
| 2 | Data types |
| 3 | Numeric constants |
| 3 | Type void |
| 4 | Type enum |
| 5 | Register variables |
| 5 | Structures |
| 5 | Newline, return, and vertical tab |
| 6 | Reserved symbols |
| 6 | Predefined symbols |
| 6 | Standard Apple Numeric Environment extensions |
| 7 | Constants |
| 8 | Expressions |
| 8 | Comparison involving a NaN |
| 8 | Functions |
| 8 | Numeric input/output |
| 9 | Numeric environment |
| 9 | About the C SANE Library |
| 9 | Programming with IEEE arithmetic |
| 10 | Pascal-compatible functions |
| 10 | Pascal-compatible function declarations |
| 11 | Pascal-compatible function definitions |
| 12 | Parameter and result data types |
| 15 | Global and external data types |
| 16 | Implementation notes |
| 16 | Size and byte alignment of variables |
| 16 | Byte ordering |
| 16 | Variable-allocation strategy |
| 17 | Types unsigned char, unsigned short and unsigned long |
| 17 | Bit fields |
| 17 | Evaluation order |
| 18 | Case statements |
| 18 | Language anachronisms |
| 18 | Assignment operators |
| 18 | Initialization |
| 19 | Structures and unions |
| 19 | Compiler limitations |

| | |
|----|---|
| 1 | Chapter 3: The Standard C Library |
| 2 | About the Standard C Library |
| 3 | Error Numbers |
| 3 | Synopsis |
| 3 | Description |
| 5 | abs — return integer absolute value |
| 6 | atof — convert ASCII string to floating-point number |
| 7 | atoi — convert string to integer |
| 8 | close — close a file descriptor |
| 9 | conv — translate characters |
| 10 | creat — create new file or rewrite existing file |
| 11 | ctype — classify characters |
| 13 | dup — duplicate an open file descriptor |
| 14 | ecvt — convert a floating-point number to a string |
| 15 | exit — terminate the current application |
| 16 | exp — exponential, log, power, square-root functions |
| 17 | faccess — named file access and control |
| 19 | fclose — close or flush a stream |
| 20 | fcntl — file control |
| 21 | ferror — stream status inquiries |
| 22 | floor, ceiling, mod, absolute value functions |
| 23 | fopen — open a buffered file stream |
| 25 | fread — binary input/output |
| 26 | frexp — manipulate parts of floating-point numbers |
| 27 | fseek — reposition a file pointer in a stream |
| 28 | getc — get a character or word from a stream |
| 29 | getenv — access exported MPW shell variables |
| 30 | gets — get a string from a stream |
| 31 | hypot — Euclidian distance function |
| 32 | ioctl — control a device |
| 34 | lseek — move read/write file pointer |
| 35 | malloc — memory allocator |
| 36 | memory — memory operations |
| 38 | onexit — install function to execute at program termination |
| 39 | open — open for reading or writing |
| 41 | printf — print formatted output |
| 45 | putc — put character or word on stream |
| 46 | puts — write a string to a stream |
| 47 | qsort — quicker sort |
| 48 | rand — a simple random-number generator |
| 49 | read — read from file |
| 50 | scanf — convert formatted input |

| | | |
|----|--------|--|
| 54 | setbuf | — assign buffering to a stream |
| 56 | setjmp | — non-local transfer of control |
| 57 | signal | — signal handling |
| 60 | sinh | — hyperbolic functions |
| 61 | stdio | — standard buffered input/output package |
| 64 | string | — string operations |
| 66 | strtol | — convert a string to a long |
| 67 | trig | — trigonometric functions |
| 68 | ungetc | — push a character block into a stream |
| 69 | unlink | — delete a named file |
| 70 | write | — write on a file |

1 Chapter 4: The Macintosh Interface Libraries

| | |
|----|--|
| 2 | About the Macintosh Interface Libraries |
| 2 | Header files |
| 2 | How the interface is implemented |
| 3 | Parameter types |
| 3 | Passing string parameters |
| 4 | Correspondence between Pascal variant records and C structs |
| 6 | InterfaceC interface to Macintosh libraries |
| 6 | Synopsis |
| 6 | Description |
| 7 | AppleTalk — AppleTalk manager |
| 12 | Controls — control manager |
| 16 | Desk — desk manager |
| 18 | Devices — device manager |
| 21 | Dialogs — dialog manager |
| 25 | DiskInit — disk initialization package |
| 26 | Disks — disk driver |
| 28 | Errors — system error handler |
| 32 | Events — Toolbox event manager |
| 36 | Files — file manager |
| 50 | FixMath — fixed-point math |
| 52 | Fonts — font manager |
| 56 | Graf3D — three-dimensional graphics routines |
| 58 | Lists — list manager package |
| 62 | Memory — memory manager |
| 65 | Menus — menu manager |
| 69 | OSEvents — operating system event manager |
| 71 | OSUtils — operating system utilities |
| 75 | Packages — package manager, disk initialization, standard file package, international utilities, binary-decimal conversion |

| | | |
|-----|-----------|---|
| 81 | Printing | — printing manager |
| 86 | Quickdraw | — graphics routines |
| 99 | Resources | — resource manager |
| 103 | Retrace | — vertical retrace manager |
| 104 | SANE | — Standard Apple Numeric Environment routines |
| 108 | Scrap | — scrap manager |
| 109 | SCSI | — SCSI manager |
| 111 | SegLoad | — Segment loader |
| 112 | Serial | — serial drivers |
| 115 | Sound | — sound drivers |
| 117 | Strings | — string conversions |
| 118 | TextEdit | — text-editing routines |
| 122 | Time | — time manager |
| 123 | ToolUtils | — Toolbox utilities |
| 126 | Types | — common defines and types |
| 127 | Windows | — window manager |

Appendixes

1 Appendix A: Calling Conventions

- 1 C calling conventions
 - 1 Parameters
 - 2 Function results
 - 2 Register conventions
- 2 Pascal-compatible calling conventions
 - 3 Parameters
 - 3 Function results
 - 4 Register conventions

1 Appendix B: Files Supplied With MPW C

- 1 C Compiler (disk C1)
- 1 C Compiler files (disk C2:CEXamples)
- 2 Libraries (disk C2:CIncludes)
- 3 Object files (disk C2:CLibraries)

1 Appendix C: The Library Index

1 Appendix D: Graf3D: Three-Dimensional Graphics

- 2 How to use Graf3D
- 2 Graf3D data types
 - 2 Point3D
 - 2 Point2D
 - 3 XfMatrix
 - 3 Port3DPtr
- 4 Graphics functions
 - 4 The InitGraf3D function
 - 4 The Open3DPort Function
 - 4 The SetPort3D Function
 - 5 The GetPort3D function
 - 5 The Move functions
 - 5 The Line functions
 - 6 The Clip3D function
 - 6 The Set Point functions
- 6 Setting up the camera
 - 6 The ViewPort function
 - 7 The LookAt function
 - 7 The ViewAngle function
- 7 The transformation matrix
 - 7 The Identity function
 - 7 The Scale function
 - 8 The Translate function
 - 8 The Pitch function
 - 8 The Yaw function
 - 8 The Roll function
 - 9 The Skew function
 - 9 The TransForm function

Appendix E: C Compiler Syntax

Glossary

Index (not included)

Figures and tables

Figure 4-1 The keyboard bit map

Table 1-1 Library object files used by MPW C

Table 1-2 Example source files used by MPW C

Table 1-4 Include-file search rules

Table 2-1 Size and range of data types

Table 2-2 Parameter and result data types

Table 2-3 Global and external data types

Table 4-1 Correspondences between variant records and structs



Preface



This manual provides information about Macintosh Programmer's Workshop (MPW) C that you need when writing applications, tools (programs that run under the MPW environment), and desk accessories and drivers for the Apple® Macintosh™ computer. It assumes you already know the C programming language.

In this manual you'll find information about compiler options, the libraries supplied with MPW C, and the differences between this implementation of C and other implementations.

What this manual contains

Here's a brief description of the contents of this manual:

- This Preface describes the manual and directs you to other reference books with information about the C language and the Macintosh programming environment.
- Chapter 1, "About MPW C," introduces MPW C and the C libraries; tells you about the sample programs and tells how to build a program; describes the C (compile) command and its options; tells you which include files to compile with and which library files to link with; and explains how to write applications, tools, and desk accessories.
- Chapter 2, "The MPW C Language," describes Apple extensions to C and clarifies aspects of the language definition as they apply to this implementation.
- Chapter 3, "The Standard C Library," documents a collection of basic routines that let you read and write files, examine and manipulate strings, perform data conversion, acquire and release memory, and perform mathematical operations.

- Chapter 4, "The Macintosh Interface Libraries," documents the interfaces between C and the Macintosh ROM and RAM routines. These interfaces enable you to write C programs that access the routines described in *Inside Macintosh*, Volumes 1-4.
- Appendix A, "Calling Conventions," defines the conventions for calling C and Pascal routines. It explains how parameters are passed to functions, how function results are returned, and how registers are saved and restored.
- Appendix B, "Files Supplied With MPW C," contains a list of all the files that are supplied with this product.
- Appendix C, "The Library Index," is a combined index of identifiers in the Standard C Library, the Macintosh Interface Libraries, and the C SANE™ Library.
- Appendix D, "Graf3D: Three-Dimensional Graphics," tells how to call three-dimensional graphics routines based on QuickDraw.
- Appendix E, "C Compiler Syntax," explains the syntax and options of the Compile command.

Other reference material you'll need

You'll need to be familiar with these additional reference materials:

- *Macintosh Programmer's Workshop Reference*. Apple Computer, Inc., 1986. This book describes the Macintosh Programmer's Workshop environment in which the C Compiler operates, including the Editor, Linker, Debugger, and other important tools.
- *The C Programming Language*. Brian W. Kernighan and Dennis M. Ritchie. Prentice-Hall, 1978. This is a standard reference book for the C language as originally defined.
- *C: A Reference Manual*. Samuel P. Harbison and Guy L. Steele, Jr. Prentice-Hall, 1984. This is a standard reference book for the C language with the Western Electric extensions used in most UNIX® operating system environments.
- *Inside Macintosh* (Volumes 1-3). Addison-Wesley, 1985. These volumes contain information you need in order to program using the Macintosh ROM and associated RAM routines; they cover windows, alert boxes, menus, graphics, and much more. Volumes 1 through 3 apply to all Macintoshes.
- *Inside Macintosh* (Volume 4). Addison-Wesley, 1986. This volume is about the 128K ROM routines available with the Macintosh Plus or Macintosh 512K enhanced. (Some of these routines, such as the Hierarchical File System routines, are also available on disk for machines that have the 64K ROM.)

- *Apple Numerics Manual*. Addison-Wesley, 1986. This book is for the programmer who wants more understanding or control of the underlying floating-point arithmetic in MPW C. It describes the Standard Apple Numeric Environment (SANE™), which includes extended-precision floating-point arithmetic as specified by IEEE Standard 754. It describes each routine in detail, including boundary conditions and exception handling, and explains how to control the floating-point environment.

Typographic and spelling conventions

This section describes the conventions used in this manual.

Language notation

This manual uses certain conventions in common with most books on C.

- C is in a monospace font:

```
int ndigit[10]
```

- Replaceable items in syntax diagrams are in italics:

```
else if (condition)  
    statement
```

Here *condition* and *statement* are expressions that are replaced by actual C expressions. The `else if` and the parentheses are C code.

Boldface

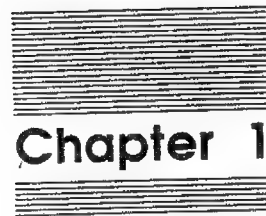
Terms that are in the glossary are set in bold type when they are defined in the text. For example, "**Standard C** is Apple's name for the de facto standard definition of C."

Spelling and capitalization

In the Standard C Library, the spelling and capitalization of identifiers is exactly as specified in the synopsis for each Standard C Library routine. Most function and parameter names are spelled entirely in lowercase. Most constant values are spelled entirely in uppercase.

In the Macintosh Interface Libraries, the spelling and capitalization of identifiers is exactly as specified in *Inside Macintosh*. Constants, variables, parameter names, fields within structures, and enumerated-type elements begin with a lowercase letter. Routines and data types begin with an uppercase letter. Letters that begin new words in English are capitalized. All other letters are lowercase. When a name contains an acronym, the case of the entire acronym is determined by the case of the first letter (for example, `GetOSEvent` and `teJustLeft`).

The SANE interface follows Standard C conventions, not *Inside Macintosh* conventions.



Chapter 1

About MPW C

MPW C is a complete implementation of the C programming language. It consists of the C Compiler, the Standard C Library, the Macintosh Interface Libraries, the C SANE Library, and example programs. The C Compiler was developed by Green Hills Software. The Standard C Library is based on the standard C library used by AT&T's UNIX System V operating system. The Standard C Library provides the Integrated Environment used by MPW tools (tools and the Integrated Environment are described later in this chapter). The Macintosh Interface Libraries provide access from C programs to the routines described in *Inside Macintosh*, as well as to the Graf3D Library. The C SANE Library supports the Standard Apple Numeric Environment described in the *Apple Numerics Manual*.

About the C Compiler

The C Programming Language by Kernighan and Ritchie provides an authoritative definition of C by its creators. However, the language has changed in several ways since the book was written, and many details of the language definition are left to the implementation to define. Therefore, the de facto standard definition of C differs somewhat from the language originally defined by Kernighan and Ritchie. This de facto standard is loosely defined by the most widely used implementation of C, the Portable C Compiler (PCC).

Standard C is the term this book uses to refer to the Berkeley 4.2 BSD VAX implementation of PCC, including the documented Western Electric extensions: type `void`, enumerated data types, and structures as function parameters and results. The MPW C Compiler is based on this de facto standard (not on the proposed ANSI standard currently under development). *C: A Reference Manual* by Harbison and Steele describes Standard C thoroughly.

MPW adds these extensions to Standard C:

- ❑ the pascal function modifier, which allows calls to and from Pascal programs and the Macintosh Interface Libraries
- ❑ the arithmetic data types `comp` and `extended`, to support the Standard Apple Numeric Environment (SANE)

Both the Western Electric extensions and the Apple extensions to C are described in Chapter 2.

About the C libraries

The following libraries are provided with MPW C:

- The **Standard C Library** (Chapter 3) is a collection of basic routines that let you read and write files, examine and manipulate strings, perform data conversion, acquire and release memory, and perform mathematical operations. This library contains functions that support MPW tools.
- The **Macintosh Interface Libraries** (Chapter 4) are a set of interfaces between C and the Macintosh ROM and RAM routines. These interfaces enable you to write C programs that access the routines described in *Inside Macintosh*, as well as the Graf3D routines.
- The **C SANE Library** provides mathematical functions and supports floating-point arithmetic. Some of these routines are called through the Standard C Library; they are documented in Chapter 3. The other SANE routines are called through the Macintosh Interface Libraries; they are documented in Chapter 4. The semantics of these routines are described in detail in the *Apple Numerics Manual*.

Within Chapters 3 and 4, the material is alphabetical by function or library name. All of the identifiers defined in the libraries are listed in a combined library index in Appendix C.

A list of the library object files used with MPW C appears in Table 1-1. The first three files, provided with the Macintosh Programmer's Workshop, are shared with other languages and are found in the {Libraries} directory. The remaining files, provided with MPW C, are used only with C and are found in the {CLibraries} directory.

Table 1-1
Library object files used by MPW C

| Libraries shared with other languages—{Libraries} directory | |
|---|--|
| Interface.o | <i>Inside Macintosh</i> libraries shared with other languages |
| ToolLibs.o | Routines normally used by tools, including the spinning cursor and error manager |
| DRVRRuntime.o | Runtime support for desk accessories and other drivers |

Table 1-1 (continued)
Library object files used by MPW C

| C Libraries—(CLibraries) directory | |
|---|---|
| CInterface.o | <i>Inside Macintosh</i> libraries specifically for C |
| StdCLib.o | Standard C Library |
| CRuntime.o | Execution starting point for applications and tools, data initialization, Quickdraw data, low-level I/O, signal handling, and built-in routines |
| CSANELib.o | SANE numerics for C |
| Math.o | Math functions, including conversions, exponential and logarithmic functions, trigonometric functions, and hyperbolic functions |

About the C examples

The {CExamples} directory contains source files for a sample application, MPW tool, and desk accessory written in C. These files are listed in Table 1-2.

Table 1-2
Example source files used by MPW C

| Source files—(CExamples) directory | |
|---|---|
| Instructions.c | Instructions for building sample programs. |
| Makefile.c | Make file for building sample programs. |
| Sample.c | Sample application. This is the sample application described in the section "A Simple Example Program" in Chapter 1 of <i>Inside Macintosh</i> , Volume 1. |
| Sample.r | Resources for sample application. |
| Count.c | Sample MPW tool. This is the source for the Count tool supplied with MPW and documented in the <i>Macintosh Programmer's Workshop Reference</i> , Chapter 9. |
| Memory.c | Sample desk accessory. The Memory desk accessory displays the current free space in the application and system heaps, the free space on the default volume, and the name of the default volume. This information is updated every five seconds. When Memory is first opened, it calls <code>_MaxMem</code> to purge memory, thus showing the upper bounds on free space in the heaps. |

Table 1-2 (continued)
Example source files used by MPW C

Source files—(CExamples) directory

| | |
|----------|---|
| Memory.r | Resources for sample desk accessory. |
| Stubs.c | Stubs for library routines not used by MPW tools. The presence of these stubs allows the Linker to reduce the size of a tool. |

The file *Instructions.c* contains step-by-step instructions for building each of the sample programs. After installing MPW and MPW C, as described in the *Macintosh Programmer's Workshop Reference*, open this file and follow the instructions.

Installing MPW C

Instructions for installing MPW on a hierarchical file system (HFS) hard disk (such as the Apple Hard Disk 20), on the Macintosh XL, and on 800K disks are found in Chapter 1 of the *Macintosh Programmer's Workshop Reference*. After installing MPW by following those instructions, install C as follows:

Hard disks

Copy file C (the Compiler) to the {MPW}Tools: folder. Copy the folders CExamples, CIncludes, and CLibraries to the {MPW} folder.

800K

Copy file C and the folders CExamples, CIncludes, and CLibraries to an 800K disk. Name the disk C. Remember to use Startup.800K as your startup file, following the instructions in the *Macintosh Programmer's Workshop Reference*. After you have examined the examples in folder CExamples, you may want to discard them to create more space on the disk.

- ❖ *Note:* You can put the Compiler, includes, and libraries in different directories, provided you change the default values of various Shell variables defined in the Startup file. You can modify the file Startup itself or, preferably, modify the file UserStartup. The following variables determine the locations of files supplied with MPW C:

| | |
|--------------|--|
| {Commands} | A comma-separated list of directories containing tools and applications. The directory containing the C Compiler should appear on this list. |
| {CIncludes} | The directory containing C include files. This should be the pathname of the CIncludes directory. |
| {CLibraries} | The directory containing C library files. This should be the pathname of the CLibraries directory. |

For more information, see “Variables” in Chapter 3 of the *Macintosh Programmer's Workshop Reference*.

Creating an application in C

An **application** is a program that can be run under the Macintosh Finder. Applications can also be run from the MPW Shell: execution of the MPW Shell is suspended and the application takes over the computer's memory and display while executing.

The code for an application is contained in 'CODE' resources in the resource fork of its file. Additional resources in the same file describe the menus, windows, dialogs, strings, and other resources used by the application. *Inside Macintosh* explains in detail how to write a Macintosh application. The following sections outline how to create an application using MPW C.

Compiling an application

To compile a C program, first start the MPW Shell application, then enter the C command in any window. Typically the command will specify options and the name of the source file to the Compiler, although neither is required. For example, the command

```
C -p Sample.c
```

compiles the source file `Sample.c`, producing the object file `Sample.c.o`. The `-p` option specifies that progress information should be written to standard output. This information will appear after the command.

If you enter the command

```
C
```

(that is, the C command without a filename), the Compiler reads from standard input—this means that the Compiler reads any text that you subsequently enter. This allows you to run the Compiler interactively. You can tell that the Compiler (rather than the MPW Shell) is reading the text you enter because the name "C" appears in the status box in the lower-left corner of the active window. Once the Compiler is running interactively, you can enter source code in any window, composing your program as you go. To terminate input, press the Command and Enter keys simultaneously. When the Compiler compiles standard input, it creates an object file named `C.o`.

- ❖ *Note:* It can be very confusing if you accidentally run the Compiler interactively. If you get the impression that the MPW Shell isn't listening to the text you enter, check the status box in the lower-left corner of the top window. It contains either the name of the Shell (if the Shell really is listening) or the name of the command that is currently executing.

A complete specification of the C command—including input, output, and diagnostic specifications, status values, and options—is found in Appendix E.

Include files

Include files (often called *header files*) are provided for both the Standard C Library and the Macintosh Interface Libraries. These files are in the {CIncludes} directory. You can determine what header files to include for a particular library function or data structure by checking the Synopsis of the appropriate section in Chapters 3 and 4.

The Compiler will search several directories for include files, until the specified file is found. The directories to search are determined by the directory containing the current input file, directories specified using the `-I` option to the Compiler, and directories specified in the Shell variable {CIncludes}.

Filenames in `#include` statements can be enclosed in either double quotation marks or angle brackets:

```
#include "MyFile.h"
#include <Types.h>
```

Normally double quotation marks are used for include files you create, and angle brackets for include files supplied with MPW. The search rules the Compiler uses in looking for the include file differ slightly in the two cases.

The form of the pathname also determines where the Compiler looks for the include file. If a *full pathname* is specified, it's the exact name of the file and no search is performed. Full pathnames contain at least one colon (:) but don't begin with a colon. If a *partial pathname* is specified, the Compiler searches several directories for the file. Partial pathnames either begin with a colon or don't contain any colons.

Table 1-3 summarizes the Compiler's include-file search rules.

Table 1-3
Include-file search rules

Full pathnames

| | |
|--|----------------------------|
| <code>#include "filename"</code> | Use the name as specified. |
| <code>#include <filename></code> | Use the name as specified. |

Partial pathnames

| | |
|----------------------------------|---|
| <code>#include "filename"</code> | Search the following directories, in this order: 1. Item |
|----------------------------------|---|

2. The directory of the source file that contains the `#include` statement
3. Directories specified by the Compiler's `-i` option, in the order specified
4. Directories specified by the Shell variable `{CIncludes}`

`#include <filename>` Search the directory listed above in steps 2, 3, and 4.

Segmentation control

A **segment** is one or more functions that can be separately loaded into memory. Your program can be written without explicit segmentation, or it can contain a number of different segments. At run time, a segment is automatically loaded by the Segment Loader when you call one of its functions. The segment is not unloaded until you explicitly unload it by calling `UnLoadSeg`. See *Inside Macintosh* for more information about the Segment Loader.

Each 'CODE' resource in the application's resource fork corresponds to a segment containing one or more functions. (The 'CODE' resource with ID 0 contains the jump table; other 'CODE' resources contain functions.) When the application is executed, each segment is automatically loaded into memory by the Segment Loader when a call is made to one of the routines in the segment. The segment is not unloaded until the application explicitly unloads it by calling `UnLoadSeg`. See the Segment Loader chapter in *Inside Macintosh*, Volume 2, for more information.

There are several ways to specify which functions are placed in which segments. This section tells how to use the Compiler's `-s` option and the `__SEG__` directive to specify segmentation. The *Macintosh Programmer's Workshop Reference* explains how to use the Link command to modify a program's segmentation.

Segmentation helps you reduce your program's runtime memory requirements. A typical segmentation algorithm divides a program into an initialization segment and a main processing segment. You can also put routines that are seldom executed—printing routines, for instance—in a separate segment that is not loaded when the program begins executing. This causes the program to be loaded faster, because the printing routines are not loaded until they are needed. If you don't specify segmentation, the Compiler puts the entire program into a segment called **Main** unless you override the default name with the `-s` compiler option.

The define directive `__SEG__` lets you specify several segments within a single source file. To assign source code to a segment, precede the code with a statement of the form

```
#define __SEG__ segmentname
```

The code following this statement is placed in the named segment until the `__SEG__` symbol is redefined or the Compiler reads the end of the source file.

- ❖ *Note:* In the `#define` directive, segment names are case sensitive. Leading and trailing spaces are significant. Unless you want the segment name to start or end with spaces, leave exactly one space between `__SEG__` and *segmentname* and no spaces after *segmentname*.

Code for a given segment does not have to be contiguous within the source file. The program may take the form

```
#define __SEG__ SegA
...function...
#define __SEG__ SegB
...function...
...function...
#define __SEG__ SegA
...function...
etc.
```

The Compiler marks each function with the name of its segment. Then the Linker collects functions for a segment from various input files and places them in a single segment in the output file.

Creating resources

Noncode resources, such as the resources that specify menus, windows, and dialogs, can be created using the Resource Editor (ResEdit) and the Resource Compiler (Rez). These tools are described in the *Macintosh Programmer's Workshop Reference*.

Linking an application

The Linker is used to combine object files from several separate compilations, together with any necessary library object files, to produce the executable code resources for a program. The Linker will either create a new resource file, containing only the code resources for your program, or replace the code resources in an existing resource file, leaving other resources, such as menus and dialogs, intact. This allows you to run the Resource Compiler either before or after running the Linker. The Linker is described in detail in the *Macintosh Programmer's Workshop Reference*.

An application written partly or totally in C should be linked with the libraries listed below.

| <i>Inside Macintosh</i> interfaces | Runtime support | Standard C Library |
|------------------------------------|------------------------|------------------------|
| {Libraries}Interface.o | {CLibraries}CRuntime.o | {CLibraries}StdCLib.o |
| {CLibraries}CInterface.o | | {CLibraries}CSANELib.o |
| | | {CLibraries}Math.o |

It's wise to link new programs with all the libraries that might be appropriate. If unnecessary files are specified, the Linker will display a warning indicating they can be removed from your build instructions.

Programs written partly in C and partly in assembly language or Pascal should be linked with the file CRuntime.o and not the file Runtime.o. The Linker will detect several duplicate entry points when linking with both the Pascal and the C libraries. All but one of these duplicates can be safely ignored: the copies of the routines are identical. The -w option in the Linker can be used to suppress the duplicate-definitions warnings.

The exception is the execution starting point. If execution is expected to begin with the C function main(), no special precautions are necessary. However, if your main program is written in assembly language or Pascal but parts of your program are written in C (and must therefore be linked with file CRuntime.o), the object file containing your main program must appear before CRuntime.o in the list of object files passed to the Linker.

A C program that calls a Pascal function or procedure requires an extern pascal declaration. (See the section "Pascal-Compatible Functions" in Chapter 2.)

Creating an MPW tool in C

A **tool** is a program that operates within the MPW Shell environment. The C Compiler, Rez, and Link are all tools. You can write your own tools in C, Pascal, or assembly language. The *Macintosh Programmer's Workshop Reference* manual describes tools and how they are created. This section contains specific information about writing tools in C.

You execute a tool by entering an MPW command. The parameters specified in the command line are passed as parameters to the function main(). The Shell variables that are exported are also passed as a parameter to main(); they can be accessed directly or by using the getenv() function from the Standard C Library. To access these parameters, declare function main() as follows:

```
main(argc, argv, envp)
    int argc;           /* number of arguments */
    char *argv[];       /* pointer to array of argument strings */
    char *envp[];       /* pointer to array of variable definitions */
```

Additional details regarding parameters to tools may be found in the *Macintosh Programmer's Workshop Reference*.

Tools have direct access to MPW Shell windows and selections. The FILE variables stdin, stdout, and stderr refer to MPW's standard input, standard output, and diagnostic output respectively. By default, Standard C Library I/O functions read standard input (text entered from the Shell) and write to standard output. Any files opened by tools, using either Standard C Library functions or *Inside Macintosh* library functions, will read and write to windows if the file specified is open in a window. The contents of the window are read or written in place of the data fork of the file. Selections in windows can also be read and written as if they were files, by adding the suffix .§ to the filename (for example, HD:MPW:Worksheet.§).

Compiling a tool

You compile a tool in exactly the same way you compile an application. The information above regarding include-file search rules, segmentation, and resources applies equally to tools and applications.

Linking a tool

The MPW Shell recognizes a tool by the type and creator. Specify the following options when linking a tool:

```
Link -t MPST -c "MPS " ...
```

The instructions above describing what library files to link with applications also apply to tools. In addition, if your tool calls any of the spinning cursor or error manager routines, you'll need to link with the following library:

```
Tool Library
    (Libraries)ToolLibs.o
```

The Integrated Environment

Tools use the Integrated Environment routines, provided as part of the Standard C Library; these routines enable a tool that calls them to run either under the MPW Shell or under the Finder. For more information about the Integrated Environment, see the *Macintosh Programmer's Workshop Reference*.

The Standard C Library routines that provide Integrated Environment facilities are close, creat, dup, faccess, fcntl, getenv, lseek, open, read, signal, unlink, and write.

These routines have C calling conventions, and their string parameters are assumed to be null-terminated C strings.

Creating a desk accessory in C

A **desk accessory** is a program run using the Apple menu. It shares its execution environment with the currently executing application. Information on writing desk accessories can be found in the Desk Manager and Device Manager chapters of *Inside Macintosh* and in the *Macintosh Programmer's Workshop Reference*. The following sections contain information specific to writing desk accessories in MPW C.

Desk accessory restrictions

A desk accessory has neither a jump table nor a global data area. Because it does not have a jump table, all of the code must be in a single segment. Either omit segmentation specification so that all your code is placed in the default Main segment, or use identical segmentation specifications for all of your functions. Use the Link command to move any library routines you use into your single segment.

Because it does not have a global data area, a desk accessory written in C may not use `extern` or `static` variables, or literal strings. Furthermore, a desk accessory cannot call library routines that require global data. Programming hints for avoiding these restrictions can be found in the *Macintosh Programmer's Workshop Reference*.

- ❖ *Note:* Apple is investigating the use of A5-based global variables in desk accessories in a future release of MPW. (Some other development systems use register A4 for this purpose. This precludes the use of library routines that depend on A5.) Currently, several Macintosh applications contain trap override or ROM hook routines that expect A5 to point to the application's globals, without saving, setting, and restoring A5 to insure that this is the case. This is incompatible with desk accessories that use A5 because calls to the ROM from the DA may end up in the application's trap override or hook code.

The DRVRRuntime library

Desk accessories have traditionally been written in assembly-language source, partly because of the peculiar resource format used by the system for desk accessories, the 'DRVVR' resource. Setting up the 'DRVVR' layout header, passing register-based procedure parameters, and coping with the nonstandard exit conventions of the driver routines have made it fairly difficult in the past for programmers not familiar with assembly language to implement desk accessories in higher-level languages.

To overcome these difficulties and simplify the task of writing a desk accessory in C, MPW provides the library DRVRRuntime.o and the resource type 'DRVW' declared in MPWTypes.r. Together they compose the driver layout header and the five entry points that set up the open, prime, status, control, and close functions of a driver. For more information about 'DRVVR' resources, see the Device Driver chapter of *Inside Macintosh*, Volume 2.

There are several advantages to using library DRVRRuntime.o in creating desk accessories. No assembly-language source is required. Each of the driver routines—DRVROpen, DRVVRPrime, DRVVRStatus, DRVVRControl, and DRVVRClose—can be written in C using standard calling conventions. The DRVRRuntime library handles desk accessory exit conventions: your routines simply return a result code.

The DRVRRuntime library consists of a main entry point that overrides the C runtime main entry point. The DRVRRuntime entry point contains driver “glue” that sets up the parameters for you, calls your routine, and performs the special exit code required by a desk accessory to return control to the system. Your routines perform the actions of the desk accessory, such as opening a window or responding to mouse clicks in it.

Desk accessory routines

Desk accessories that use the library DRVRRuntime must contain the five functions DRVROpen, DRVVRPrime, DRVVRStatus, DRVVRControl, and DRVVRClose. All of these functions have the same parameter and result types. They are declared as Pascal-compatible functions so that the library DRVRRuntime may be used for writing desk accessories in C, Pascal, and assembly language. Each of these five routines should be declared as follows:

```
pascal OSErr DRVROpen(ct1PB,dCtl)
    CntrlParam *ct1PB;
    DCtrlPtr dCtl;
{
    ...your code here...
    return(resultCode);
}
```

Types `CntrlParam` and `DctlPtr` are defined in the file `Devices.h`. Type `OSErr` is a short and is defined in `Types.h`. Details on what each of these functions should do are found in the *Macintosh Programmer's Workshop Reference*.

Linking a desk accessory

A desk accessory written in C must be linked with both `DRVRRuntime.o` and `CRuntime.o`. `DRVRRuntime.o` must precede `CRuntime.o` in the list of object files passed to the Linker.

```
Desk Accessories
(Libraries)DRVRRuntime.o
```




Chapter 2



The MPW C Language

The information provided in this chapter supplements *The C Programming Language* by Kernighan and Ritchie.

Where Kernighan and Ritchie's language definition leaves choices to the implementers, this chapter describes how these aspects of C have been implemented on the Macintosh. Where Apple has modified or extended their language definition, this chapter documents the changes.

Language definition

This section describes the MPW C language, including language extensions such as type void, type enum, the SANE data types, and calling Pascal-compatible functions.

Variable names

The Compiler does not place a limit on the length of local variable names. Global variable names and function names are limited to 63 characters by the object-module format. Therefore, different function names whose first 63 characters are identical will be treated as different functions by the Compiler but will be treated as the same function by the Linker.

Data types

Table 2-1 lists the arithmetic and pointer types available in MPW C and shows the number of bits allocated for variables of these types. Types int and long, which are identical in this implementation, represent 32-bit integers. Pointers also require 32 bits. Enumerated types are allocated 8, 16, or 32 bits, depending on the range of the enumerated literal values.

Table 2-1
Size and range of data types

| Data type | Bits | Description |
|----------------|--------------|---|
| char | 8 | range -128 to 127 |
| unsigned char | 8 | range 0 to 255 |
| short | 16 | range -32,768 to 32,767 |
| unsigned short | 16 | range 0 to 65,535 |
| int | 32 | range -2,147,483,648 to 2,147,483,647 |
| unsigned int | 32 | range 0 to 4,294,967,295 |
| long | 32 | range -2,147,483,648 to 2,147,483,647 |
| unsigned long | 32 | range 0 to 4,294,967,295 |
| enum | 8, 16, or 32 | depends on the range of the enumerated literals |
| * | 32 | pointer types |
| float | 32 | IEEE single-precision floating point |
| double | 64 | IEEE double-precision floating point |
| comp | 64 | SANE signed integral values |
| extended | 80 | IEEE extended-precision floating point |

Note: Type short int is equivalent to short, and type long int is equivalent to long.

Numeric constants

Integer constants in the range of long are treated as type long. Integer constants outside the range of long are treated as type extended, not type unsigned long as you might expect. For example, the initialization statement

```
long i = 4000000000;
```

is incorrect because 4,000,000,000, being too big for a long, is interpreted as an extended value. However, the initialization statement

```
unsigned long i = 4000000000;
```

is correct because 4,000,000,000 is within the range of unsigned long.

Type void

The **void** keyword tells the Compiler that the function being declared does not return a value. Calls to functions of type void may not be used in expressions, where a value is required. (See "Pascal-Compatible Functions" later in this chapter.)

Type void can also be used in a cast to explicitly discard the return value of a function call, as in

```
(void) printf("Hello");
```

Type enum

Type **enum** is a type analogous to the enumerated types of Pascal. Its syntax is similar to that of the struct and union declarations:

enum-specifier:

```
enum { enum-list }  
    enum identifier { enum-list }  
    enum identifier
```

enum-list:

```
    enumeration-declaration  
    enumeration-declaration, enum-list
```

enumeration-declaration:

```
    identifier  
    identifier = constant-expression
```

The first identifier in *enum-specifier*, like the structure tag in a struct specifier, names a particular enumeration. For example,

```
enum color {chartreuse, burgundy, claret, winedark};  
enum color *cp, col;
```

These declarations make `color` the enumeration tag of a type describing various colors and then declares `cp` as a pointer to an object of that type and `col` as an object of that type.

The identifiers in *enum-list* are declared as constants and may appear wherever constants are required. If no *enumeration-declarations* with a *constant-expression* appear, the values of the constants begin at 0 and increase by 1 as the declaration is read from left to right. An *enumeration-declaration* with a *constant-expression* gives the associated identifier the value indicated; subsequent identifiers continue the progression by 1 from the assigned value.

Enumeration constants must be unique. They are drawn from the set of ordinary identifiers, unlike field names in structures. Objects of a given enumerated type have a type distinct from objects of all other types.

Enumerated types are allocated the amount of space required by the smallest predefined type that allows representation of all the literal values specified by the enumeration. The predefined types are `char` and unsigned `char` (8 bits), `short` and unsigned `short` (16 bits), and `int` and unsigned `int` (32 bits). The `-z6` compiler option overrides the allocation algorithm and forces the Compiler to allocate 32 bits for all enumerated data types.

Important

If you use the `-z6` option, your enumerated types will not match data structures used in the Macintosh ROM.

Register variables

The Compiler allocates automatic variables in registers whenever possible. Register variables will be assigned to registers before other automatic variables. Enumeration, character, integer, and pointer variables qualify for register allocation unless their address is taken with the & operator. Floating-point variables are not allocated to registers.

Several data and address registers are available for use as automatic variables. The exact number depends on the calling conventions used. The number of variables allocated to registers may exceed the total number of registers. Several variables whose useful lifetimes do not overlap may be assigned to the same register. Often all of the eligible variables within a function will reside in registers, rather than on the stack.

Structures

Structures may be assigned, passed as parameters, and returned as function results. Other operators, such as equality comparison, are not available for structures.

The left and right sides of a structure assignment must have the same type. Similarly, actual and formal parameters must have identical types.

Important

Functions that return structures are not reentrant. If an interrupt occurs during the return sequence and the same function is called during the interrupt, the value returned from the first call may be incorrect. The problem can occur only in the presence of interrupts. Recursive calls are quite safe.

The newline, carriage-return, and vertical-tab characters

The **newline** character in output advances the print position or cursor to the left margin on the next line. In C notation, newline is represented by `\n`. The character code for `\n` in this implementation is the ASCII value (13) for the carriage-return character, not the ASCII value (10) for the linefeed character, as in most other implementations.

In C notation, the **carriage-return** character is represented by `\r`. The character code for `\r` in this implementation is the standard ASCII value (13) for the carriage-return character. Therefore `\r` and `\n` are equal in MPW C, although they are not equal in most other implementations.

The vertical-tab character (ASCII 31), represented by `\v`, is not meaningfully interpreted in Editor windows or by TextEdit.

Reserved symbols

`__LINE__` is a reserved preprocessor symbol whose value is the current line number within the current source file.

`__FILE__` is a reserved preprocessor symbol whose value is a character string consisting of the current filename.

`__SEG__` is a reserved preprocessor symbol that overrides the segment name associated with functions that follow; it remains in effect until the next `__SEG__` directive. The default segment name, Main, may also be overridden with the `-s` compiler option.

`__LINE__`, `__FILE__`, and `__SEG__` begin and end with two underscore characters.

Predefined symbols

The following symbols are predefined for use in conditional compilation:

```
MC68000
mc68000
m68k
ghs
macintosh
```

Each of the predefined symbols has the value 1, as if a statement of this form had appeared at the beginning of the source code:

```
#define MC68000 1
```

The `-u` compiler option lets you undefine any of the predefined symbols.

Standard Apple Numeric Environment extensions

MPW C has built-in support for the Standard Apple Numeric Environment (SANE). SANE is introduced in this section and documented completely in the *Apple Numerics Manual*. Chapters 3 and 4 of this book include the interface to the C SANE Library, which supplements the SANE facilities built into the MPW C language. Because the C SANE Library provides a number of functions, such as `exp`, that are found in the Standard C Library, the interfaces to these functions appear in Chapter 3, "The Standard C Library." Additional functions appear in Chapter 4, "The Macintosh Interface Libraries."

The MPW C language and the C SANE Library together implement the IEEE Standard for Binary Floating-Point Arithmetic (754). SANE adds a data type to the IEEE types and provides basic functions for application development. MPW C recognizes the SANE data types (float, double, comp, and extended), uses SANE for all floating-point operations and conversions, and correctly handles NaNs (Not-a-Number) and infinities in comparisons and in ASCII-binary conversions. Furthermore, source programs written using only Standard C float and double types and Standard C operations are portable—they will compile and run in MPW C as well as in other C implementations.

Much of SANE is provided through the runtime library CSANELib.o and its include file SANE.h. However, to use extended-precision arithmetic efficiently and effectively and to handle IEEE NaNs and infinities, some extensions to Standard C are required, including the use of the extended data type.

The change from double to extended as the basic floating-point type is the most important difference from Standard C. Because C was originally developed on the DEC PDP-11, the PDP-11 architecture is reflected in Standard C in the use of float and double as floating-point types, with double as the basic type: floating-point expressions are evaluated to double, anonymous variables are double, and floating-point parameters and function results are passed as doubles. However, the low-level SANE arithmetic (as well as the floating-point chips Intel 8087, Motorola 68881, and Zilog Z8070) evaluates arithmetic operations to the range and precision of an 80-bit extended type. Thus, extended naturally replaces PDP-11 double as the basic arithmetic type for computing purposes. The types float (IEEE single), double, and comp serve as space-saving storage types, just as float does in conventional C.

The IEEE Standard specifies two special representations for its floating-point formats: NaNs (Not-a-Number) and infinities. MPW C expands the syntax for I/O to accommodate NaNs and infinities, and includes the treatment of NaNs in relationals as required by the IEEE Standard.

The SANE extensions to Standard C are backward compatible: programs written using only float and double floating-point types and Standard C operations compile and run without modification. SANE does not affect integer arithmetic.

Constants

Numeric constants that include floating-point syntax—a point (.) or an exponent field—or that lie outside the range of type long are of type extended. Decimal-to-binary conversion for numeric constants is done at compile time and hence is governed by the default numeric environment (see “Numeric Environment” in this chapter).

Expressions

The SANE types—float, double, comp, and extended—can be mixed in expressions with one another and with integer types in the same manner that float and double can in Standard C. An expression consisting solely of a SANE-type variable, constant, or function is of type extended. An expression formed by subexpressions and an arithmetic operation is of type extended if either of its subexpressions is. Extended-type expressions are evaluated using extended-precision SANE arithmetic, with conversions to type extended generated automatically as needed. Parentheses in extended-type expressions are honored. Initialization of external and static variables is done at compile time; all other evaluation of extended-type expressions is done at run time.

Comparisons involving a NaN

The result of a comparison involving a NaN operand is **unordered**. The usual set of comparison results—less than (<), greater than (>), and equal to (==)—is expanded to include unordered. For example, the negation of “*a* less than *b*” is not “*a* greater than or equal to *b*” but “(*a* greater than or equal to *b*) OR (*a* and *b* unordered).”

Functions

A numeric actual parameter passed by value is an expression and hence is of extended or an integer type. All extended-type arguments are passed as extended. Similarly, all results of functions declared float, double, comp, or extended are returned as extended.

Numeric input/output

In addition to the usual syntax accepted for numeric input, the Standard C Library function `scanf` recognizes the string “INF” as infinity and the string “NaN” as a NaN. “NaN” may be followed by parentheses, which may contain an integer (a code indicating the NaN’s origin). “INF” and “NaN” are optionally preceded by a sign and are case insensitive. The `scanf` specifiers for SANE types extend Standard C as follows: conversion characters `f`, `e`, and `g` indicate type float; `lf`, `le`, and `lg` indicate type double; `mf`, `me`, and `mg` indicate type comp; and `ne`, `nf`, and `ng` indicate type extended.

The Standard C Library function `printf` writes infinities as “INF” and NaNs as “NaN(*ddd*)”, where *ddd* is the NaN code. “INF” and “NaN(*ddd*)” may be preceded by a minus sign.

Numeric environment

The numeric environment comprises the rounding direction, rounding precision, halt enables, and exception flags. IEEE Standard defaults—rounding to nearest, rounding to extended precision, and all halts disabled—are in effect for compile-time arithmetic (including decimal-to-binary conversion). Each program begins with these defaults and with all exception flags clear. Functions for managing the environment are included in the C SANE Library (CSANELib.o). The Compiler, in optimizing, will not change any part of the numeric environment (including the exception-flag setting, which is a side effect of arithmetic operations).

About the C SANE Library

The C SANE Library provides the basic tools for developing a wide range of applications. It includes the following:

- ☐ logarithmic, exponential, and trigonometric functions
- ☐ financial functions
- ☐ random-number generation
- ☐ binary-decimal conversion
- ☐ numeric scanning and formatting
- ☐ environment control
- ☐ other functions required or recommended by the IEEE Standard

See Chapters 3 and 4 for the interface to the C SANE Library.

Programming with IEEE arithmetic

MPW C's automatic use of the extended type produces results that are generally better than those of most other C systems. For example, extended precision yields more accuracy, and extended range avoids unnecessary underflow and overflow of intermediate results. You can further exploit the extended type by declaring all floating-point temporary variables to be of type extended. This is both time-efficient and space-efficient, because it reduces the number of automatic conversions between types. External data should be stored in one of the three smaller SANE types (float, double, or comp), not only for economy but also because the extended format may vary between SANE implementations. As a general rule, use float, double, or comp data as program input; extended arithmetic for computations; and float, double, or comp data as program output.

In many instances, IEEE arithmetic allows simpler algorithms than were possible without IEEE arithmetic. The handling of infinities enlarges the domain of some formulas. For example, $1+1/x^2$ computes correctly even if x^2 overflows. While running with halts disabled (the default), a program will never crash because of a floating-point exception. Hence by monitoring exception flags, a program can test for exceptional cases after the fact. The alternative of screening out bad input is often infeasible, and sometimes impossible.

Pascal-compatible functions

The function-calling conventions used by MPW C and Pascal differ in the order of parameters on the stack, the type coercions applied to parameters, the location of the return result, and the number of scratch registers. C has been extended to allow function calls between these languages. The `pascal` specifier in a function declaration or definition indicates a **Pascal-compatible function**.

Pascal-compatible function declarations

A function or procedure written in Pascal (or written in C or assembly language following Pascal calling conventions) can be called from C. This section tells you how to declare Pascal-compatible functions in C. Appendix A describes both the C calling conventions and the Pascal-compatible calling conventions.

A Pascal-compatible external function declaration begins with the `pascal` specifier; contains the usual type specifiers, function name, and parameter list; and must contain declarations for the parameters, followed by the word `extern`. Pascal-compatible function declarations are external declarations—that is, they may not appear within functions or compound statements. Parameters whose declarations are omitted are assumed to be type `int`.

Here is an example of a procedure, `DrawText`, as defined in Pascal:

```
PROCEDURE DrawText(textBuf: Ptr; firstByte, byteCount: Integer);
```

Here is the corresponding C function declaration for the `DrawText` procedure. Note that the declaration contains the `void` keyword because in Pascal a procedure does not return a value.

```
pascal void DrawText(textBuf, firstByte, byteCount)
    Ptr textBuf;
    short firstByte, byteCount;
extern;
```

Pascal-compatible function declarations are used in the Macintosh Interface Libraries to allow C programs to directly call Macintosh library routines that use Pascal calling conventions. The word `extern` may be followed by a constant, which is interpreted as a 16-bit 68000 instruction that replaces the usual subroutine call (JSR) instruction in the calling sequence. This allows direct traps to the Macintosh ROM. For example,

```
pascal void OpenPort(port)
    GrafPtr port;
extern 0xA86F;
```

Pascal-compatible function definitions

A C function definition (the actual function), like a function declaration, can also be preceded by the pascal specifier. The function is then given Pascal-compatible calling conventions by the Compiler. For example, the following C function can be called from Pascal:

```
pascal void MyText(byteCount, textAddr, numer, denom)
    short byteCount;
    Ptr textAddr;
    Point numer, denom;
{
    ...
}
```

The corresponding Pascal function declaration is

```
PROCEDURE MyText(bytecount: integer; textAddr: Ptr; numer, denom: Point);
```

For compatibility with Pascal and assembly language, the Compiler converts the names of Pascal-compatible functions to uppercase before writing them to the object file. When they are called in C programs, these routines should be capitalized exactly as they were declared in C. Pascal-compatible functions whose names differ only in their capitalization will become duplicate declarations when their names are converted to uppercase by the Compiler; therefore such names should be avoided.

Parameter and result data types

C and Pascal support different data types. Therefore when writing a Pascal-compatible function declaration in C, a translation of the parameter types and function-result type (from Pascal to C) is required. Often this translation is obvious, but some cases are surprising. For example, Pascal passes type `char` as a 16-bit value.

Table 2-2 summarizes this translation. Find the Pascal parameter or result type in the first column. Use the equivalent C type found in the second column when declaring the function in C. Comments in the table point out unusual cases that may require special attention.

Table 2-2
Parameter and result data types

| Pascal data type | C equivalent | Comments |
|---|------------------------|--|
| <code>boolean</code> | <code>boolean</code> | Boolean is defined in file <code>Types.h</code> as enum <code>{false, true}</code> : |
| <code>VAR boolean</code> | <code>boolean *</code> | In C, <code>false</code> is zero and <code>true</code> is often considered nonzero. |
| <code>boolean result</code> | <code>boolean</code> | In Pascal, <code>false</code> is zero and <code>true</code> is one. |
| enumeration type (<code><128</code> or <code>>255</code> literals) | <code>enum</code> | Use identical ordering of the enumeration literals. |
| enumeration type (<code>128</code> to <code>255</code> literals) | <code>short</code> | Pascal passes enumerations with <code>128</code> or more literals as words. |
| <code>VAR</code> enumeration type (<code><128</code> or <code>>255</code> literals) | <code>enum *</code> | |
| <code>VAR</code> enumeration type (<code>128</code> to <code>255</code> literals) | <code>short *</code> | |
| enumeration-type result (<code><128</code> or <code>>255</code> literals) | <code>enum</code> | |
| enumeration-type result (<code>128</code> to <code>255</code> literals) | <code>short</code> | |
| <code>char</code> | <code>short</code> | Pascal passes <code>char</code> parameter as 16-bit values. |

| | | |
|---------------------|-----------------|---|
| var char | short * | Pascal stores unpacked char types as 16-bit values. |
| char result | short | |
| integer | short | 16-bit signed values. |
| VAR integer | short * | |
| integer result | short | |
| longint | int or long | 32-bit signed values. |
| VAR longint | int * or long * | |
| longint result | int or long | |
| real | extended * | Pascal passes real parameters as extended by address. |
| VAR real | float * | |
| real result | float | Pascal returns real results by value. |
| double | extended * | Pascal passes double parameters as extended by address. |
| VAR double | double * | |
| double result | double | The caller supplies the address of the double result. |
| comp | extended * | Pascal passes comp parameters as extended by address. |
| VAR comp | comp * | |
| comp result | comp | The caller supplies the address of the comp result. |
| extended | extended * | Pascal passes extended parameters by address. |
| VAR extended | extended * | |
| extended result | extended | The caller supplies the address of the extended result. |
| pointer type | pointer | 32-bit addresses. |
| VAR pointer type | pointer * | |
| pointer-type result | pointer | |

| | | |
|---------------------------------|-------------|---|
| ARRAY (1 or 2 bytes) | short | Pascal passes small arrays by value. |
| ARRAY (3 or 4 bytes) | int or long | |
| ARRAY (5 or more bytes) | array | Pascal passes larger arrays by address. |
| VAR ARRAY | array | |
| ARRAY result | — | C does not allow arrays as results. |
| RECORD (1 to 4 bytes) | struct | Pascal passes small records by value. |
| RECORD (5 or more bytes) | struct * | Pascal passes larger records by address. |
| VAR RECORD (any size) | struct * | |
| record result (1 or 2 bytes) | struct | Pascal returns small records by value. |
| RECORD result (3 or 4 bytes) | struct | |
| RECORD result (5 or more bytes) | struct | The caller supplies the address of the record result. |
| SET (1 to 7 elements) | char | Pascal passes sets with 1 to 7 elements as bytes. |
| SET (8 to 16 elements) | short | Pascal passes sets with 8 to 16 elements as words. |
| SET (≥ 17 elements) | struct | Pascal also passes larger sets by value. |
| VAR SET (1 to 7 elements) | char * | |
| VAR SET (8 to 16 elements) | short * | |
| VAR SET (≥ 17 elements) | struct * | |
| SET result (1 to 7 elements) | char | Pascal returns small sets by value. |

| | | |
|----------------------------------|--------|---|
| SET result (8 to 16 elements) | short | |
| SET result (≥17 elements) | struct | The caller supplies the address of the set result. |

❖ *Note:* The C `struct` type and the Pascal `record` type do not exactly correspond, as C lacks an equivalent to the Pascal `variant record` type. You can see the relationship by comparing the data structures in the Files section of Chapter 4 with those in the File Manager chapter of *Inside Macintosh*, Volume 4.

Global and external data types

When a C program and a Pascal program use the same global or external variables, they must use the corresponding data types. These are shown in Table 2-3. The first column shows the Pascal type. The second column shows the equivalent C types (sometimes there are more than one).

Table 2-3
Global and external data types

| Pascal data type | C equivalent | Comment |
|------------------|--|--------------------------------------|
| boolean | boolean | Defined in file <code>Types.h</code> |
| enumeration type | enum | |
| char | short | |
| -128..127 | char | |
| 0..255 | short | |
| integer | short or unsigned short | |
| longint | int, unsigned int, long, or unsigned long | |
| real | float | |
| double | double | |
| comp | comp | |
| extended | extended | |
| pointer type | pointer | |
| STRING | Str255 | Defined in file <code>Types.h</code> |

Implementation notes

A number of details in every C language definition are left to the discretion of the implementers. Most programs do not rely on these details and therefore yield the same results on the various implementations. However, if you want to write programs that will run under more than one implementation, you need to know the specific semantics of each C compiler. This section explains several areas of the language definition that are specific to MPW C.

Size and byte alignment of variables

Variables of type `char`, and variables of type `enum` that require only a single byte of memory, are aligned in memory on byte boundaries. All other types are aligned on word boundaries: that is, they have even memory addresses. Note that `struct` types are aligned on word boundaries, and that the smallest `struct` (a `struct` with one `char` in it) takes up two bytes. The packing of `enum` types is described in the "Type Enum" section in this chapter. Signed and unsigned types have the same size and alignment.

Byte ordering

The Macintosh's microprocessor, the Motorola 68000, stores the least significant byte of a short or long integer at the highest memory address. This byte ordering is also used on IBM System/370 and Zilog Z8000 processors. The 6502 family of processors used in the Apple II family of computers, the Intel 8086 family, the PDP-11 family, the DEC VAX, and the National Semiconductor NS16000 store the least significant byte at the lowest address. Programs that rely on the order of the bytes within short and long integers will not work correctly on both classes of machines.

Variable-allocation strategy

The MPW C Compiler optimizes allocation of variables in various ways. Automatic variables (locals) are allocated in registers whenever possible. Static and global variables are not necessarily allocated in the order in which they are specified. (However, the order of fields within structures is preserved.) Static variables may be allocated as if they were automatic if their values are always set before being referenced. Automatic and static variables that are never used may not be allocated at all. Programs should not rely on the Compiler's allocation algorithms.

Types unsigned char, unsigned short, and unsigned long

Types unsigned char, unsigned short, and unsigned long are supported by the MPW C Compiler and by many implementations of PCC, although they are not required by the basic C language definition. The VAX implementation of PCC and the MPW C Compiler differ in the way they evaluate expressions involving these types. For example, the negation operator subtracts an unsigned short from 2^{16} under PCC and from 2^{32} under MPW C.

Bit fields

MPW C provides bit fields that are unsigned, as do many if not all 68000 versions of PCC. However, VAX implementations of C may support signed bit fields. In the following example, implementations using unsigned bit fields will set `i` to 3; implementations using signed bit fields will set `i` to -1:

```
struct {int field:2;} x;  
int i;  
  
x.field = 3;  
i = x.field;
```

The `-x55` compiler option causes the Compiler to treat bit fields as signed rather than unsigned.

Evaluation order

MPW C does not define the evaluation order of certain expressions. Expressions with side effects, such as function calls and the `++` and `--` operators, may yield different results on different machines or with different compilers. Specifically, when a variable is modified as a side effect of an expression's evaluation and the variable is also used at another point in the same expression, the value used may be either the value before modification or the value after modification.

Programs should not rely on the order of evaluation in these situations. The function call

```
f(i, i++)
```

is an example of an expression whose value is undefined.

Case statements

Some implementations of C, including PCC, allow cases of a `switch` statement to be nested within compound statements. MPW C considers this an error. The following `switch` statement compiles using PCC but generates an error message using the MPW C Compiler. The error is that `case 2:` is within the `if` statement.

```
switch (i) {
    case 1:
        if (j) {
            case 2:
                i = 3;
        }
}
```

Language anachronisms

Several constructs formerly considered part of the C language are now considered anachronisms. When you specify the `-z84` compiler option, anachronistic constructs are compiled and flagged with a warning message. Otherwise they are considered invalid. The anachronisms involve assignment operators, initialization statements, and references to structures and unions.

Assignment operators

The `=op` form of assignment operators is not supported. Alternate interpretations are accepted without warning. In particular,

| | | |
|--------------------------|-------------------|----------------------------|
| <code>x =- 5;</code> | is interpreted as | <code>x = (-5);</code> |
| <code>x =* 5;</code> | is interpreted as | <code>x = (*5);</code> |
| <code>x =& p;</code> | is interpreted as | <code>x = (&p);</code> |

Initialization

The equal sign that introduces an initializer must be present. The anachronism

```
int i 1;
```

is considered an error.

Structures and unions

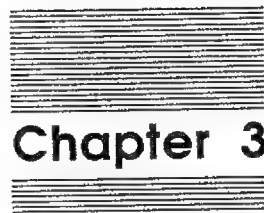
References to members of structures and unions must be to the appropriate structure or union. For example, the reference `a.b` is illegal if `b` is not a member of `a`. References to components of nested structures and unions must be fully qualified (that is, all intermediate levels of the reference must be specified).

The names of structure and union members do not conflict with the names of ordinary variables in the same scope. Furthermore, a particular member name may be used in several structures and unions in the same scope.

Compiler limitations

The total size of all declared global variables, static variables, and string constants cannot exceed 32K bytes. Allocate large global arrays on the heap in order to avoid exceeding this limit.

The size of the largest function you can compile in MPW C is limited by the memory available for the Compiler's internal data structures. The problem can be reduced by eliminating unnecessary include files, reducing the number of global declarations, compiling large functions separately, and rewriting large functions as two or more smaller functions. You can also make more memory available to the Compiler by compiling without the debugger installed.



Chapter 3

The Standard C Library

About the Standard Library

This chapter describes the Standard C Library provided with MPW C. The Standard C Library (Chapter 3) is a collection of basic routines that let you read and write files, examine and manipulate strings, perform data conversion, acquire and release memory, and perform mathematical operations.

The chapter begins with an introduction to the error-number conventions used in the Standard C Library, followed by the library functions and macros arranged alphabetically by header. For example, both the `fread` and `fwrite` macros are found under the `fread` header. All of the function names and other identifiers used in Standard C Library routines are listed in Appendix C, "The Library Index."

❖ *Note:* Remember that identifiers in C are case sensitive and should be spelled exactly as shown in the synopsis.

The library routines under each header are documented as follows:

- *Synopsis* shows the code you need to add to your program when using these library routines and files you need to include at compile time.
- *Description* discusses the library routines and their input and output.
- *Diagnostics* describes error conditions.
- *Return value* describes the values returned by the routines.
- *Note* contains remarks.
- *Warning* gives cautions.
- *See also* provides the names of other library routines or sections in this chapter related to the ones described in the current document.

Error numbers

Synopsis

```
#include <ErrNo.h>

extern int errno;
extern short MacOSErr;
```

Description

Many of the Standard C Library functions have one or more possible error returns. An otherwise meaningless return value, usually -1, indicates an error condition; see descriptions of individual functions for details. The external variable `errno` also provides an error number. Variable `errno` is not cleared on successful calls, so it should be tested only if the return value indicates an error.

The error name appears in brackets following the text in a library function description; for example,

"The next attempt to write a nonzero number of bytes will signal an error. [ENOSPC]"

Not all possible error numbers are listed for each library function because many errors are possible for most of the calls. Some UNIX operating system error numbers do not apply to Macintosh and are not documented in this manual. Some calls go to the Macintosh ROM and return a value in `MacOSErr` as well as the value in `errno`.

Here is a list of the error numbers and their names as defined in the `<ErrNo.h>` file.

- 2 **ENOENT** *No such file or directory*
This error occurs when a file whose filename is specified does not exist or when one of the directories in a pathname does not exist.
- 3 **ENOSRRC** *Resource not found*
A required resource was not found. This error applies to `faccess` calls that return tab, font, or print record information.
- 5 **EIO** *I/O error*
Some physical I/O error has occurred. This error may in some cases be signaled on a call following the one to which it actually applies.
- 6 **ENXIO** *No such device or address*
I/O on a special file refers to a subdevice that does not exist, or the I/O is beyond the limits of the device. This error may also occur when, for example, no disk is present in a drive.
- 7 **E2BIG** *Insufficient space for return argument*
The data to be returned is too large for the space allocated to receive it.
- 9 **EBADF** *Bad file number*
Either a file descriptor does not refer to an open file, or a read (or write) request is made to a file that is open only for writing (or reading).

- 12 **ENOMEM** *Not enough space*
The system ran out of memory while the library call was executing.
- 13 **EACCES** *Permission denied*
An attempt was made to access a file in a way forbidden by the protection system.
- 14 **GFAULT** *Illegal filename*
A filename or volume name was too long or otherwise illegal.
- 17 **EEXIST** *File exists*
An existing file was mentioned in an inappropriate context; for example, `open(file, O_CREAT+O_EXCL)`.
- 19 **ENODEV** *No such device*
An attempt was made to apply an inappropriate system call to a device; for example, read a write-only device.
- 20 **ENOTDIR** *Not a directory*
An object that is not a directory was specified where a directory is required; for example, in a path prefix.
- 21 **EISDIR** *Is a directory*
An attempt was made to write on a directory.
- 22 **EINVAL** *Invalid parameter*
Some invalid parameter was provided to a library function.
- 23 **ENFILE** *File table overflow*
The system's table of open files is full, so temporarily a call to open cannot be accepted.
- 24 **EMFILE** *Too many open files*
The system cannot allocate memory to record another open file.
- 28 **ENOSPC** *No space left on device*
During a write to an ordinary file, there is no free space left on the device.
- 29 **ESPIPE** *Illegal seek*
An lseek was issued incorrectly.
- 30 **EROFS** *Read-only file system*
An attempt to modify a file or directory was made on a device mounted for read-only access.
- 31 **EMLINK** *Too many links*
An attempt to delete an open file was made.

Note

Calls that interface to the Macintosh I/O system (such as `open`, `close`, `read`, `write`, and `ioctl`) can set the external variable `MacOSErr` as well as `errno` on errors. This section documents the `errno` values. The equivalent Macintosh ROM error-return values set in `MacOSErr` are documented in the Errors manual page in Chapter 4 and in the System Error Handler chapter of *Inside Macintosh*.

abs—return integer absolute value

Synopsis

```
int abs(i)  
int i;
```

Description

Function `abs` returns the absolute value of `i`.

Note

The absolute value of the negative integer with the largest magnitude is undefined.

See also

`floor`

atof—convert ASCII string to floating-point number

Synopsis

```
extended atof(str)
char *str;
```

Description

Function `atof` converts a character string pointed to by `str` to an extended-precision floating-point number. The first unrecognized character ends the conversion. Function `atof` recognizes an optional string of white-space characters (spaces or tabs), then an optional sign, then a string of digits optionally containing a decimal point, then an optional *e* or *E* followed by an optionally signed integer. If the string begins with an unrecognized character, `atof` returns a NaN.

Function `atof` recognizes "INF" as infinity and "NaN" (optionally followed by parentheses that may contain a string of digits) as a NaN, with NaN code given by the string of digits. Case is ignored in the infinity and NaN string.

Diagnostics

Function `atof` honors the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by SANE.

See also

`scanf`, `str2dec`, `dec2num`
Apple Numerics Manual

atoi—convert string to integer

Synopsis

```
int atoi(str)
char *str;
long atol(str)
char *str;
```

Description

The character string `str` is scanned up to the first nondigit character other than an optional leading minus sign (-). Leading white-space characters (spaces and tabs) are ignored.

Return value

Function `atoi` returns as an integer the decimal value represented by `str`. Function `atol` returns as a long integer the decimal value represented by `str`. On the Macintosh, these functions are equivalent because `int` and `long` are the same size.

Note

Overflow conditions are ignored.
A plus sign (+) is considered a nondigit character.

See also

`atof`, `scanf`, `strtol`

close—close a file descriptor

| | |
|--------------------|---|
| Synopsis | <pre>int close(fildes) int fildes;</pre> |
| Description | Parameter <code>fildes</code> is a file descriptor obtained from an <code>open</code> , <code>creat</code> , <code>dup</code> , or <code>fcntl</code> call. Function <code>close</code> closes the file descriptor indicated by <code>fildes</code> . Function <code>close</code> fails if <code>fildes</code> is not a valid open file descriptor. [EBADF] |
| Diagnostics | Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and <code>errno</code> is set to indicate the error. |
| See also | <code>creat</code> , <code>dup</code> , <code>fcntl</code> , <code>open</code> |

conv—translate characters

Synopsis

```
#include <CType.h>

int toupper(c)
    int    c;
int tolower(c)
    int    c;
int _toupper(c)
    int    c;
int _tolower(c)
    int    c;
int toascii(c)
    int    c;
```

Description

Functions `toupper` and `tolower` have as their domain the set of ASCII characters (0 through 127) and the constant EOF (-1). If parameter `c` to `toupper` represents a lowercase letter, the result is the corresponding uppercase letter. If parameter `c` to `tolower` represents an uppercase letter, the result is the corresponding lowercase letter. All other parameters in the domain are returned unchanged.

Macros `_toupper` and `_tolower` produce the same results as functions `toupper` and `tolower` but have restricted domains and are faster. Macro `_toupper` requires a lowercase letter as its parameter; its result is the corresponding uppercase letter. Macro `_tolower` requires an uppercase letter as its parameter; its result is the corresponding lowercase letter. Parameters outside the domain cause undefined results.

Function `toascii` converts `c` by clearing all bits that are not part of a standard ASCII character. It is used for compatibility with other systems.

Note

These routines do not support the Macintosh extended character set (with values greater than 0x7F). For values outside the stated domain, the result is undefined.

See also

`ctype`, `getc`

creat—create a new file or rewrite an existing file

Synopsis

```
int creat(filename)
    char *filename;
```

Description

Function `creat` creates a new file or prepares to rewrite an existing file, `filename`. If the file exists, the length of its data fork is set to 0.

Function `creat(filename)` is equivalent to

```
open(filename, O_WRONLY+O_TRUNC+O_CREAT)
```

Upon successful completion, a nonnegative integer (the file descriptor) is returned and the file is open for writing. The file pointer is set to the beginning of the file. A maximum of about 30 files may be open at a given time; the actual maximum depends upon the current system environment.

Return value

Upon successful completion, a nonnegative integer (the file descriptor) is returned. Otherwise, a value of `-1` is returned and `errno` is set to indicate the error.

Note

Other implementations of `creat` specify a second parameter, `mode`. This version ignores any second parameter.

See also

`close`, `open`

ctype—classify characters

Synopsis

```
#include <CType.h>

int isalpha(c)
    int c;
int isalpha(c)
    int c;
int isupper(c)
    int c;
int islower(c)
    int c;
int isdigit(c)
    int c;
int isxdigit(c)
    int c;
int isalnum(c)
    int c;
int isspace(c)
    int c;
int ispunct(c)
    int c;
int isprint(c)
    int c;
int isgraph(c)
    int c;
int isctrl(c)
    int c;
int isascii(c)
    int c;
```

Description

These macros classify character-coded integer values by table lookup, returning nonzero for true, zero for false. Macro `isascii` is defined on all integer values; the rest are defined only where `isascii` is true and on the single non-ASCII value EOF (-1).

| Macro | Returns true if |
|-----------------------|---|
| <code>isascii</code> | <code>c</code> is an ASCII character code lower than octal 0200. |
| <code>isalpha</code> | <code>c</code> is a letter [A-Z] or [a-z]. |
| <code>isupper</code> | <code>c</code> is an uppercase letter [A-Z]. |
| <code>islower</code> | <code>c</code> is a lowercase letter [a-z]. |
| <code>isdigit</code> | <code>c</code> is a digit [0-9]. |
| <code>isxdigit</code> | <code>c</code> is a hexadecimal digit [0-9], [A-F], or [a-f]. |
| <code>isalnum</code> | <code>c</code> is alphanumeric (letter or digit). |
| <code>isspace</code> | <code>c</code> is a space, tab, return, new line, vertical tab, or form feed. |

| | |
|----------------------|---|
| <code>ispunct</code> | <code>c</code> is a punctuation character (neither control nor alphanumeric). |
| <code>isprint</code> | <code>c</code> is a printing character, space (octal 040) through tilde (octal 0176). |
| <code>isgraph</code> | <code>c</code> is a printing character, similar to <code>isprint</code> except false for space. |
| <code>isctrl</code> | <code>c</code> is a delete character (octal 0177) or an ordinary control character (less than octal 040). |

Warning If `c` is not in the domain of the function, the result is undefined.

Note These macros do not support the Macintosh extended character set. For values outside the domain, the result is undefined.

dup—duplicate an open file descriptor

Synopsis

```
int dup(fildes)
int    fildes;
```

Description

Parameter `fildes` is a file descriptor obtained from an `open`, `creat`, `dup`, or `fcntl` call. The new file descriptor returned by `dup` is the lowest one available.

The function call `dup(fildes)` is equivalent to

```
fcntl(fildes, F_DUPFD, 0)
```

Function `dup` fails if parameter `fildes` is not a valid open file descriptor. [EBADF]

Return value

Upon successful completion, a nonnegative integer (the file descriptor) is returned. Otherwise, a value of `-1` is returned and `errno` is set to indicate the error.

See also

`close`, `fcntl`, `open`

ecvt—convert a floating-point number to a string

Synopsis

```
char *ecvt(value, ndigit, decpt, sign)
    extended value;
    int ndigit, *decpt, *sign;
char *fcvt(value, ndigit, decpt, sign)
    extended value;
    int ndigit, *decpt, *sign;
```

Description

Function `ecvt` converts `value` to a null-terminated string of `ndigit` digits and returns a pointer to this string as the function result. The low-order digit is rounded.

The decimal point is not included in the returned string. The position of the decimal point is indicated by `decpt`, which indirectly stores the position of the decimal point relative to the returned string. If the `int` pointed to by `decpt` is negative, the decimal point lies to the left of the returned string. For example, if the string is "12345" and `decpt` points to an `int` of 3, the value of the string is 123.45; if `decpt` points to -3, the value of the string is .00012345.

If the sign of the converted value is negative, the `int` pointed to by `sign` is nonzero; otherwise it is zero.

Function `fcvt` provides fixed-point output in the style of Fortran F-format output. Function `fcvt` differs from `ecvt` in its interpretation of `ndigit`:

- In `fcvt`, `ndigit` specifies the number of digits to the right of the decimal point.
- In `ecvt`, `ndigit` specifies the number of digits in the string.

Note

The string pointed to by the function result is static data whose contents are overwritten by each call. To preserve the value, copy it before calling the function again.

See also

`printf`, `num2dec`, `dec2str`
Apple Numerics Manual

exit—terminate the current application

Synopsis

```
void exit(status)
    int    status;
void _exit(status)
    int    status;
```

Description

Functions `exit` and `_exit` close open file descriptors and terminate the application or tool. Here is the order in which `exit` performs its duties:

1. It executes all exit procedures in reverse order of their installation by `onexit`, including the exit procedures for the Standard I/O package if Standard I/O routines were used. All buffered files are flushed and closed.
2. It closes all open files that were opened with `open` or `fopen`.
3. If the program is a tool running under the MPW Shell, the `exit` function returns status and control to the MPW Shell by placing a return value in the lower three bytes of status and terminating the application.

Function `_exit` circumvents the exit procedures described in step 1 above. Use `_exit` instead of `exit` to abort your program when you are uncertain about the integrity of the data space.

Return value

The main program is a function that returns an integer. The return value of `main` is interpreted by the MPW Shell as the program status. When you call `exit` or `_exit`, the status parameter is returned to the MPW Shell as the return value for the application's main function: 0 for normal execution or a small positive value for errors (typically 1..3). Main programs that return to the Shell without setting status to an integer value appear to be returning a random status.

There is no return from `exit` or `_exit`.

Note

Functions `exit` and `_exit` do not close files you opened with calls to the I/O routines documented in *Inside Macintosh*.

Don't call `exit` or `_exit` from a desk accessory.

See also

`onexit`, `stdio`

exp—exponential, logarithm, power, square-root functions

Synopsis

```
#include <Math.h>

extended exp(x)
    extended x;
extended log(x)
    extended x;
extended log10(x)
    extended x;
extended pow(x, y)
    extended x, y;
extended sqrt(x)
    extended x;
```

Description

Function `exp(x)` returns e^x , where e is the natural logarithm base.

Function `log(x)` returns the natural logarithm of x , $\log_e x$.

Function `log10(x)` returns the base-10 logarithm of x , $\log_{10} x$.

Function `pow(x, y)` returns x^y .

Function `sqrt(x)` returns the square root of x .

For special cases, these functions return a NaN or signed infinity as appropriate.

Diagnostics

These functions honor the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by SANE.

See also

hypot, sinh
Apple Numerics Manual

faccess—named file access and control

Synopsis

```
#include <FCntl.h>

int faccess(filename, cmd, arg)
    char    *filename;
    unsigned int cmd;
    long    *arg;
```

Description

Function `faccess` provides access to control and status information for named files. (Compare function `ioctl`, which provides different control and status information for open files.)

Parameter `cmd` must be set to one of the constants in the following list to indicate what operation is to be performed on the file. As noted in the list, some calls to `faccess` also require the `arg` parameter, usually as a `long` or as a pointer to a `long`.

The following commands are available to all programs:

| Value of cmd | Description |
|-----------------------|---|
| <code>F_DELETE</code> | Deletes the named file, or returns an error if the file is open. Parameter <code>arg</code> is ignored. |
| <code>F_RENAME</code> | Renames the named file. Parameter <code>arg</code> is a pointer to a string containing the new name. |

The following commands can be used only by a program running as an MPW tool:

| Value of cmd | Description |
|--------------------------|---|
| <code>F_GTABINFO</code> | Gets the tab offset for the MPW text file <code>filename</code> . The tab offset is stored in the long integer pointed to by <code>arg</code> . The tab offset is expressed as an integer number of spaces. The width of the space character in the current font determines the actual distance of the tab offset. |
| <code>F_STABINFO</code> | Sets the tab offset for the MPW text file <code>filename</code> . The tab offset is specified as a long value in <code>arg</code> . |
| <code>F_GFONTINFO</code> | Gets the font number and font size for an MPW text file <code>filename</code> . The font number is stored in the high-order half of the long pointed to by <code>arg</code> ; the font size is stored in the low-order half of the same long. |
| <code>F_SFONTINFO</code> | Sets the font number and font size for the MPW text file, <code>filename</code> . The font number is specified in the high-order half of <code>arg</code> ; the font size is specified in the low-order half of <code>arg</code> . |

| | |
|--|---|
| <code>F_GPRINTREC</code> | Gets a print record <code>TPrint</code> for an MPW text file, <code>filename</code> ; <code>arg</code> is a handle to the print record. |
| <code>F__SPRINTREC</code> | Sets a print record for the MPW text file <code>filename</code> ; <code>arg</code> is a handle to the print record. |
| <code>F_OPEN</code> | Reserved for operating system use. |
| <code>F_GTABINFO</code> and <code>F_GFONTINFO</code> | pass <code>arg</code> as a pointer to a long; |
| <code>F_STABINFO</code> and <code>F_SFONTINFO</code> | pass <code>arg</code> as a long value; and |
| <code>F_GPRINTREC</code> and <code>F__SPRINTREC</code> | pass <code>arg</code> as a handle to a print record. |

Return value Upon successful completion, `faccess` returns a nonnegative value, usually 0. If the device for the named file cannot perform the requested command, `faccess` returns -1 and `errno` is set to indicate the error.

If the requested resource for `F_GTABINFO`, `F_GFONTINFO`, or `F_GPRINTREC` does not exist for the named file, default values are stored and the function returns a value greater than 0.

Note Before calling `faccess` with `F_GPRINTREC` or `F__SPRINTREC`, the Printing Manager must be initialized and the print record handle `THPrint` must be allocated. The font size must be 9, 10, 12, 14, 18, or 24; the font number must be 0 or a positive integer. The following sequence must be used with these print command values:

```
res = CurResFile();
PRClose();
UseResFile(res);
PROpen(); /* do whatever, including call faccess print commands */
PRClose();
UseResFile(res);
```

See also `ioctl`, `unlink`

fclose—close or flush a stream

Synopsis

```
#include <StdIO.h>

int fclose(stream)
    FILE *stream;
int fflush (stream)
    FILE *stream;
```

Description

Function `fclose` closes a file that was opened by `fopen`, `freopen`, or `fdopen`. Function `fclose` causes any buffered data for `stream` to be written out, and the buffer (if one was allocated by the system) is released; `fclose` then calls `close` to close the file descriptor associated with `stream`. The value of the parameter `stream` cannot be used unless reassigned with `fopen`, `fdopen`, or `freopen`.

Function `fclose` fails if the file descriptor associated with `stream` is already closed. [ENOENT]

Function `fclose` is performed automatically for all open `FILE` streams upon calling `exit`.

Function `fflush` causes any buffered data for `stream` to be written out; `stream` remains open.

Return value

These functions return 0 for success or EOF if an error was detected (such as trying to write to a file that has not been opened for writing).

See also

`close`, `exit`, `fopen`, `setbuf`

fcntl—file control

Synopsis

```
#include <fcntl.h>

int fcntl(fildes, cmd, arg)
    int    fildes;
    unsigned int cmd;
    int     arg;
```

Description

Function `fcntl` is used for duplicating file descriptors. A file remains open until all of its file descriptors are closed.

Parameter `fildes` is an open file descriptor obtained from an `open`, `creat`, `dup`, or `fcntl` call. Parameter `cmd` takes the value `F_DUPFD`, which tells `fcntl` to return the lowest numbered available file descriptor greater than or equal to `arg`. Normally `arg` is greater than or equal to 3, to avoid obtaining the standard file descriptors 0, 1, and 2. Function `fcntl` returns a new file descriptor that points to the same open file as `fildes`. The new file descriptor has the same access mode (read, write, or read/write) and file pointer as `fildes`. Any I/O operation changes the file pointer for all file descriptors that share it.

Function `fcntl` fails if one or more of the following are true:

- Parameter `fildes` is not a valid open file descriptor. [EBADF]
- Parameter `arg` is negative or greater than the highest allowable file descriptor. [EINVAL]

Return value

Upon successful completion, the value returned is a new file descriptor. Otherwise, a value of `-1` is returned and `errno` is set to indicate the error.

Note

The `F_GETFD`, `F_SETFD`, `F_GETFL`, and `F_SETFL` commands of `fcntl` are not supported on the Macintosh.

See also

`close`, `dup`, `open`

error—stream status inquiries

Synopsis

```
#include <StdIO.h>

int feof(stream)
    FILE *stream;
int ferror(stream)
    FILE *stream;
void clearerr(stream)
    FILE *stream;
int fileno(stream)
    FILE *stream;
```

Description

Macro `feof` returns nonzero when end of file has previously been detected reading the named input stream; otherwise, it returns zero.

Macro `ferror` returns nonzero when an I/O error has previously occurred reading from or writing to the named stream; otherwise, it returns zero.

Macro `clearerr` resets the error indicator and end-of-file indicator to zero on the named stream.

Macro `fileno` returns the integer file descriptor associated with the named stream; see `open`.

See also

`open`, `fopen`

floor—floor, ceiling, mod, absolute value functions

Synopsis

```
#include <Math.h>

extended floor(x)
    extended x;
extended ceil(x)
    extended x;
extended fmod(x, y)
    extended x, y;
extended fabs(x)
    extended x;
```

Description

Function `floor(x)` returns the largest integer (as an extended-precision number) not greater than `x`.

Function `ceil(x)` returns the smallest integer not less than `x`.

Whenever possible, `fmod(x, y)` returns the number f with the same sign as `x`, such that $x = iy + f$ for some integer i , and $|f| < |y|$. If `y` is 0, `fmod` returns a NaN.

Function `fabs(x)` returns $|x|$, the absolute value of `x`.

See also

`abs`, `rint`, `setround`
Apple Numerics Manual

fopen—open a buffered file stream

Synopsis

```
#include <StdIO.h>

FILE *fopen(filename, type)
    char *filename, *type;
FILE *freopen(filename, type, stream)
    char *filename, *type;
    FILE *stream;
FILE *fdopen(fildes, type)
    int fildes;
    char *type;
```

Description

Function `fopen` opens the file named by `filename` and associates a stream with it. Function `fopen` returns a pointer to the `FILE` structure associated with the stream. Parameter `filename` points to a character string that contains the name of the file to be opened.

The value of `type` should be one of the values in the first column in the following table. The headings "Open Mode Used" and "Description" explain how `type` is used. For more information, see `open`.

| Value | Open mode used | Description |
|-------|--------------------------|---|
| r | O_RDONLY | Open for reading only. |
| w | O_WRONLY+O_CREAT+O_TRUNC | Truncate or create for writing. |
| a | O_WRONLY+O_APPEND | Append: open for writing at end of file, or create for writing. |
| r+ | O_RDWR | Open for update (reading and writing). |
| w+ | O_RDWR+O_CREAT+O_TRUNC | Truncate or create for update. |
| a+ | O_RDWR+O_CREAT+O_APPEND | Append: open or create for update at end of file. |

Function `freopen` substitutes the named file for the open stream. The original stream is closed, regardless of whether the open ultimately succeeds. Function `freopen` returns a pointer to the `FILE` structure associated with `stream`. Function `freopen` is typically used to attach the previously opened streams associated with `stdin`, `stdout`, and `stderr` to other files.

Function `fdopen` associates a stream with a file descriptor by formatting a file structure from the file descriptor. Thus, `fdopen` can be used to access the file descriptors returned by `open`, `creat`, `dup`, or `fcntl`. (These calls return file descriptors, not pointers to a `FILE` structure.) The type of stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening `fseek` or `rewind`, and input may not be directly followed by output without an intervening `fseek`, `rewind`, or an input operation that encounters end of file.

When a file is opened for append (that is, when `ttype` is `a` or `a+`), it is impossible to overwrite information already in the file. Function `fseek` may be used to reposition the file pointer to any position in the file, but when output is written to the file the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output.

Return values On success, functions `fopen`, `freopen`, and `fdopen` return a valid file pointer. On failure, `NULL` is returned.

The maximum number of open `FILE` streams is 20.

Note The parameter `type` must have one of the values in the first column in the table; do not use values intended for `open`, such as `O_RDONLY`.

See also `open`, `fclose`, `fseek`

fread—binary input/output

Synopsis

```
#include <StdIO.h>

int fread(ptr, size, nitems, stream)
    char *ptr;
    int size, nitems;
    FILE *stream;
int fwrite(ptr, size, nitems, stream)
    char *ptr;
    int size, nitems;
    FILE *stream;
```

Description

Function `fread` copies `nitems` items of data from the named input stream into an array beginning at `ptr`. An item of data is a sequence of `size` bytes (not necessarily terminated by a null byte). Function `fread` stops appending bytes if an end of file or error condition is encountered while reading `stream` or if `nitems` items have been read. Function `fread` leaves the file pointer in `stream` pointing to the byte following the last byte read.

Function `fwrite` writes at most `nitems` items of data to the named output stream from the array pointed to by `ptr`. An item is a sequence of `size` bytes. Function `fwrite` stops writing when it has written `nitems` items of data or if an error condition is encountered on `stream`. Function `fwrite` does not change the contents of the array pointed to by `ptr`.

The parameter `size` is typically

`sizeof(*ptr)`

where `sizeof` specifies the length of an item pointed to by `ptr`. If `ptr` points to a data type other than `char`, it should be cast into a pointer to `char`.

Return values Functions `fread` and `fwrite` return the number of items read or written. If `nitems` is 0 or negative, no characters are read or written and 0 is returned by both `fread` and `fwrite`.

See also `fopen`, `getc`, `gets`, `printf`, `putc`, `puts`, `read`, `scanf`, `stdio`, `write`

frexp—manipulate parts of floating-point numbers

Synopsis

```
extended frexp(value, eptr)
    extended value;
    int *eptr;
extended ldexp(value, exp)
    extended value;
    int exp;
extended modf(value, iptr)
    extended value, *iptr;
```

Description

Every nonzero number can be written uniquely as $x \cdot 2^n$, where the mantissa (fraction) x is in the range $0.5 \leq |x| < 1.0$ and the exponent n is an integer. Function `frexp` returns the mantissa of an extended value and stores the exponent indirectly in the location pointed to by `eptr`. Note that the mantissa here differs from the significand described in the *Apple Numerics Manual*, whose normal values are in the range $1.0 \leq |x| < 2.0$.

Function `ldexp` returns the quantity $value \cdot 2^{exp}$.

Function `modf` returns the signed fractional part of `value` and stores the integral part indirectly in the location pointed to by `iptr`.

Diagnostics

Function `ldexp` honors the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by SANE.

See also

`logb`, `scalb`
Apple Numerics Manual

fseek—reposition a file pointer in a stream

Synopsis

```
#include <StdIO.h>

int fseek(stream, offset, ptrname)
    FILE *stream;
    long offset;
    int ptrname;
void rewind(stream)
    FILE *stream;
long ftell(stream)
    FILE *stream;
```

Description

Function `fseek` sets the position of the next input or output operation on the stream. The new position is offset bytes from the beginning, the current position, or the end of the file, when the value of `ptrname` is 0, 1, or 2, respectively. If `ptrname` is 1 or 2, offset may be negative.

The call

```
rewind(stream)
```

is equivalent to

```
fseek(stream, 0L, 0)
```

except that no value is returned.

Functions `fseek` and `rewind` undo any effects of `ungetc`.

After `fseek` or `rewind`, the next operation on a file opened for update may be either input or output.

Function `ftell` returns the offset of the current byte relative to the beginning of the file associated with the named stream.

Diagnostics

Function `fseek` returns nonzero for improper seeks; otherwise it returns zero. An example of an improper seek is an `fseek` before the beginning of, or past the end of, the file.

See also

`lseek`, `fopen`, `ungetc`

getc—get a character or a word from a stream

Synopsis

```
#include <StdIO.h>

int getc(stream)
    FILE *stream;
int getchar()
int fgetc(stream)
    FILE *stream;
int getw(stream)
    FILE *stream;
```

Description

Macro `getc` returns the next character from the named input stream. It also moves the file pointer, if defined, ahead one character in `stream`. Macro `getc` cannot be used if a function is necessary; for example, you cannot have a function pointer point to it. Macro `getc` returns the integer EOF on end of file or error.

Macro `getchar` returns the next character from the standard input stream, `stdin`.

Function `fgetc` produces the same result as macro `getc`; function `fgetc` runs more slowly than macro `getc` but takes less space per invocation. You can also have a pointer to `fgetc` but not to `getc`.

Function `getw` returns the next `int` (that is, four bytes) from the named input stream so that the order of bytes in the stream corresponds to the order of bytes in memory. Function `getw` returns the constant EOF upon end of file or error. Because EOF is a valid integer value, `feof` and `ferror` should be used to check the success of `getw`. Function `getw` increments the associated file pointer, if defined, to point to the next `int`. Function `getw` assumes no special alignment in the file.

Return values These calls return data from the stream, or the integer constant EOF (−1) at end of file or upon an error.

Note Because it is implemented as a macro, `getc` treats a stream parameter with side effects incorrectly. In particular,

```
getc(*f++)
```

doesn't work as you would expect. Instead use

```
fgetc(*f++)
```

See also `ferror`, `fopen`, `fread`, `gets`, `scanf`, `stdio`

getenv—access exported MPW Shell variables

Synopsis

```
char *getenv(varname)
char  *varname;
```

Description

The **environment** is the set of exported variables provided by the MPW Shell. Function `getenv` provides access to variables in this set. (See the Variables section in Chapter 3 of the *Macintosh Programmer's Workshop Reference* for the list of standard exported Shell variables.)

Function `getenv` searches the environment for a Shell variable with the name specified by `varname` and returns a pointer to the character string containing its value. The null pointer is returned if the Shell variable is not defined or has not been exported. The Shell-variable name search is case insensitive.

Return value

Upon successful completion, a pointer to the value of `varname` is returned. If the Shell variable is not defined or not exported, the function returns the null pointer.

For standalone applications, which do not run under the MPW Shell, `getenv` always returns the null pointer.

Note

The environment can also be accessed by means of a parameter to the C main-entry-point function `main` if the main procedure is declared as

```
main(argc, argv, envp)
```

The `envp` array represents the set of MPW Shell variables that have been made available to tools by means of the MPW Export command. The *i*th `envp` entry has the form

```
envp[i] = "varname\0varvalue\0";
```

The last `envp` entry is the null pointer.

If you use `envp` to search the environment, be sure to use case-insensitive string comparisons.

Warning

Function `getenv` returns a pointer to the place in memory where a copy of the MPW Shell variable resides. Do not modify the value of a Shell variable in such a way as to increase its length.

gets—get a string from a stream

Synopsis

```
#include <StdIO.h>

char *gets(str)
    char *str;
char *fgets(str, maxlen, stream)
    char *str;
    int maxlen;
    FILE *stream;
```

Description

Function `gets` reads characters from the standard input stream `stdin` into the array pointed to by `str` until a newline character is read or an end-of-file condition is encountered. The newline character is discarded, and the string is terminated with a null (`\0`) character. (For more information about newline, see "The Newline, Carriage-Return, and Vertical-Tab Characters" in Chapter 2.)

Function `fgets` reads characters from `stream` into the array pointed to by `str` until `maxlen-1` characters are read, a newline character is read and transferred to `str`, or an end-of-file condition is encountered. The string is then terminated with a null character.

Return values

If end of file is encountered and no characters have been read, no characters are transferred to `str` and `NULL` is returned. If a read error occurs, `NULL` is returned. Otherwise `str` is returned. (A read error will occur, for example, if you attempt to use these functions on a file that has not been opened for reading.)

Note

The array pointed to by `str` is assumed to be large enough; overflow is not checked. The function `gets` omits the newline character in the string; `fgets` leaves it in.

See also

`ferror`, `fopen`, `fread`, `getc`, `scanf`, `stdio`

hypot—Euclidean distance function

Synopsis

```
#include <Math.h>

extended hypot (x, y)
    extended x, y;
```

Description

Function hypot returns
 $\text{sqrt}(x * x + y * y)$
taking precautions against unwarranted overflows.

Diagnostics

Function hypot honors the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by SANE.

See also

exp
Apple Numerics Manual

ioctl—control a device

Synopsis

```
#include <IOctl.h>

int ioctl(fildes, cmd, arg)
    int    fildes;
    unsigned int cmd;
    long   *arg;
```

Description

Function `ioctl` communicates with a file's device handler by sending control information, requesting status information, or both. Parameter `cmd` indicates which device-specific operations `ioctl` must perform. Here are the control values:

| Value of cmd | Description |
|-----------------------------|---|
| <code>FIOINTERACTIVE</code> | Function <code>ioctl</code> returns 0 if the device is interactive; if not, it returns -1 and <code>errno</code> is set to <code>EINVAL</code> . Parameter <code>arg</code> is ignored. |
| <code>FIOBUFSIZE</code> | Function <code>ioctl</code> returns, in bytes, the optimal buffer size for this device; the buffer size is returned in a long pointed to by <code>arg</code> . If the device has no default buffer size, <code>ioctl</code> returns -1 and <code>errno</code> is set to <code>EINVAL</code> . |
| <code>FIOFNAME</code> | Function <code>ioctl</code> stores the filename associated with <code>fildes</code> in a character array 256 characters in size pointed to by <code>arg</code> . It returns -1 if the filename exceeds 255 characters [<code>E2BIG</code>]. |
| <code>FIOREFNUM</code> | Function <code>ioctl</code> returns the Macintosh file reference number associated with <code>fildes</code> ; the reference number is returned in the short pointed to by <code>arg</code> . If the <code>fildes</code> is not open on a Macintosh file (such as the console device), <code>ioctl</code> returns -1. |
| <code>FIOSETEOF</code> | Function <code>ioctl</code> sets the logical end of file specified in the long parameter <code>arg</code> . The value of <code>arg</code> is the new size of the file, in bytes. This command can be used to reduce or increase the size of the open file. The current file pointer is not affected unless the file size is set below it. |
| <code>TIOFLUSH</code> | Used only for the console device and other terminal devices. Function <code>ioctl</code> returns -1 if <code>fildes</code> is not a terminal device. <code>TIOFLUSH</code> tells the device handler to throw away unread terminal input. Parameter <code>arg</code> is ignored. |

The following two functions are used only for the console device when running standalone:

| | |
|-----------------|--|
| TIOGPORT | Function <code>ioctl</code> returns the console <code>GrafPort</code> in a <code>GrafPtr</code> pointed to by <code>arg</code> . |
| TIOSPORT | Function <code>ioctl</code> sets the console <code>GrafPort</code> to the value specified by the <code>GrafPtr</code> <code>arg</code> . Subsequent writes to the console will display on this <code>GrafPort</code> . |

Function `ioctl` fails if one or both of the following conditions exist:

- ☐ File descriptor `fildes` is not valid or is not open. [EBADF]
- ☐ Parameters `cmd` or `arg` are not valid for the device handler associated with `fildes`. [EINVAL]

Diagnostics If an error has occurred, a value of `-1` is returned and `errno` is set to indicate the error.

Note For `cmd` values `FIOINTERACTIVE` and `FIOBUFSIZE`, a function return of `-1` is a meaningful response, not an error. For `FIOINTERACTIVE`, `errno` is set to `EINVAL` for devices that are not interactive. For `FIOBUFSIZE`, `errno` is set to `EINVAL` for devices that have no default buffering.

The `cmd` values `FIOLSEEK` and `FIODUPFD` are reserved for operating system use.

If you set the console `GrafPort` with `TIOSPORT`, do not deallocate the storage for that port; the console device is written to by the `exit` function as your application terminates.

Warning `FIOREFNUM` lets you do Macintosh I/O operations such as `Allocate` that are not available through `ioctl`. Do not close or modify the file pointer using the reference number.

See also `fcntl`

lseek—move read/write file pointer

Synopsis

```
long lseek(fildes, offset, whence)
    int    fildes;
    long   offset;
    int    whence;
```

Description

A file descriptor, `fildes`, is returned from a call to `creat`, `dup`, `fcntl`, or `open`. Function `lseek` sets the file pointer associated with `fildes` as follows:

- If `whence` is 0, the pointer is set to `offset` bytes.
- If `whence` is 1, the pointer is set to its current location plus `offset`.
- If `whence` is 2, the pointer is set to the size of the file plus `offset`.
- If `whence` is 1 or 2, the value of `offset` may be negative.

Upon successful completion, the file pointer value as measured in bytes from the beginning of the file is returned.

The file pointer remains unchanged and `lseek` fails if one or more of the following are true:

- File descriptor `fildes` is not open. [EBADF]
- Parameter `whence` is not 0, 1, or 2. [EINVAL]
- The resulting file pointer would point past end of file. [ESPIPE]
- The resulting file pointer would point before beginning of file. [EINVAL]

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

Return value

Upon successful completion, a nonnegative long integer indicating the file-pointer value is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

Note

In previous versions of the Standard C Library, `tell(fildes)` was a function that returned the current file position. It is equivalent to the call

```
lseek(fildes, 0L, 1)
```

Warning

Function `lseek` has no effect on a file opened with the `O_APPEND` flag because the next write to the file always repositions the file pointer to the end before writing."

See also

`fseek`, `open`

malloc—memory allocator

Synopsis

```
char *malloc(size)
    unsigned int size;
void free(ptr)
    char *ptr;
char *realloc(ptr, size)
    char *ptr;
    unsigned int size;
char *calloc(nelem, elsize)
    unsigned int nelem, elsize;
void cfree(ptr, nelem, elsize)
    char *ptr;
    unsigned int nelem, elsize;
```

Description

Functions `malloc` and `free` provide a simple general-purpose memory allocation package. The storage area expands as necessary when `malloc` is called.

Function `malloc` allocates the first sufficiently large contiguous free space it finds and returns a pointer to a block of at least `size` bytes suitably aligned for any use. It calls `NewPtr` (see *Inside Macintosh*) to get more memory from the system when there is no suitable space already free.

Function `free` takes a parameter that is a pointer to a block previously allocated by `malloc`. If its size is greater than 2K bytes, it is returned to the system using `DisposePtr`. Blocks smaller than that are cached by `malloc` for further allocation by `malloc` only. Undefined results occur if the space assigned by `malloc` is overrun or if a random value is passed to `free`.

Function `realloc` changes the size of the block pointed to by `ptr` to `size` bytes and returns a pointer to the (possibly moved) block. The contents are unchanged up to the lesser of the new and old sizes. If no free block of `size` bytes is available in the storage area, `realloc` asks `malloc` to enlarge the storage area by `size` bytes and then moves the data to the new space. If `ptr` is `NULL`, `realloc` is equivalent to `malloc`.

Function `calloc` allocates space for an array of `nelem` elements of size `elsize`. The space is initialized to zeros.

Function `cfree`, like `free`, frees memory allocated by `calloc`; `cfree` is included for compatibility with other systems. Parameters `nelems` and `elsize` are ignored.

Diagnostics

Functions `malloc`, `realloc`, and `calloc` return a null pointer if there is no available memory or if the storage area has been detectably corrupted by storing outside the bounds of a block. When this happens, the block pointed to by `ptr` may have been destroyed.

memory—memory operations

Synopsis

```
char *memcpy(dest, source, c, n)
    char *dest, *source;
    int c, n;
char *memchr(source, c, n)
    char *source;
    int c, n;
int memcmp(a, b, n)
    char *a, *b;
    int n;
char *memcpy(dest, source, n)
    char *dest, *source;
    int n;
char *memset(dest, c, n)
    char *dest;
    char c;
    int n;
```

Description

These functions operate efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

Function `memcpy` copies characters from memory area `source` into `dest`, stopping after the first occurrence of character `c` has been copied or after `n` characters have been copied, whichever comes first. It returns either a pointer to the character after the copy of `c` in `dest` or a null pointer if `c` was not found in the first `n` characters of `source`.

Function `memchr` returns either a pointer to the first occurrence of character `c` in the first `n` characters of memory area `source` or a null pointer if `c` does not occur.

Function `memcmp` compares its parameters, `a` and `b`, looking at the first `n` characters only. It returns an integer less than, equal to, or greater than 0, depending on whether `a` is less than, equal to, or greater than `b` in the ASCII collating sequence.

Function `memcpy` copies `n` characters from memory area `source` to `dest`. It returns `dest`.

Function `memset` sets the first `n` characters in memory area `dest` to the value of character `c`. It returns `dest`.

Warning

Overlapping moves yield unexpected results.

Function `memcmp` uses signed arithmetic when comparing its parameters. The sign of the result will be incorrect for characters with values greater than 0x7F in the Macintosh extended character set.

See also `BlockMove, string`

onexit—install a function to be executed at program termination

Synopsis

```
int onexit(func);  
void (*func)();
```

Description

Function `onexit` installs the `exit` function pointed to by `func` by adding it to a list. The list is initially empty. A list entry is added whenever `onexit` is called. Function `exit` calls the functions in the list in the reverse order in which they were added. To ensure that buffers are flushed at program termination, the Standard I/O package adds its cleanup function to the list the first time it allocates a buffer. Each function in the list is called without parameters either at program termination or when `exit` is called.

The number of user-supplied exit functions is limited to seven.

Diagnostics

The function returns a nonzero value if the installation succeeds.

Note

A call to `__exit` circumvents user exit procedures installed by `onexit`.

Warning

If a function is installed more than once, the behavior is undefined.

See also

`exit`, `stdio`

open—open for reading or writing

Synopsis

```
#include <fcntl.h>
int open(filename, oflag)
    char *filename;
    int oflag;
```

Description

Parameter `filename` is a disk file, window, selection, or pseudofile. (See the section “Pseudo-Filenames” in Chapter 3 of the *Macintosh Programmer's Workshop Reference* for more information.) Function `open` opens a file descriptor for the named file and sets the file-status flags according to the value of `oflag`. The value of `oflag` is constructed by OR-ing flag settings; for example,

```
filides = open("MyFile", O_WRONLY|O_CREAT|O_TRUNC);
```

To construct `oflag`, first select one of the following access modes:

- ☐ `O_RDONLY` Open for reading only.
- ☐ `O_WRONLY` Open for writing only.
- ☐ `O_RDWR` Open for reading and writing.

Then optionally add one or more of these modifiers:

- ☐ `O_APPEND` The file pointer is set to the end of the file before each write.
- ☐ `O_CREAT` If the file does not exist, it is created.
- ☐ `O_TRUNC` If the file exists, its length is truncated to 0; the mode and owner are unchanged.
- ☐ `O_RSRC` The file's resource fork is opened. (Normally, the data fork is opened.)

The following setting is valid only if `O_CREAT` is also specified:

- ☐ `O_EXCL` Function `open` fails if the file exists.

Upon successful completion, a nonnegative integer (the file descriptor) is returned. The file pointer used to mark the current position within the file is set to the beginning of the file.

The named file is opened unless one or more of the following are true:

- ☐ `O_CREAT` is not set and the named file does not exist. [ENOENT]
- ☐ More than about 30 file descriptors are currently open. The actual limit varies according to runtime conditions. [EMFILE]
- ☐ `O_CREAT` and `O_EXCL` are set and the named file exists. [EEXIST]

Return value

Upon successful completion, a nonnegative integer (the file descriptor) is returned. Otherwise, a value of `-1` is returned and `errno` is set to indicate the error.

See also `close, creat, lseek, read, write`

printf—print formatted output

Synopsis

```
#include <StdIO.h>

int printf(format [ , arg ] ... )
    char *format;
int fprintf(stream, format [ , arg ] ... )
    FILE *stream;
    char *format;
int sprintf(str, format [ , arg ] ... )
    char *str, format;
```

Description

Function `printf` places formatted output on the standard output stream `stdout`. Function `fprintf` places formatted output on the named output stream `stream`. Function `sprintf` places formatted output, followed by the null character (`\0`), into the character array pointed to by `str`; it's your responsibility to ensure that enough room is available. Each function returns the number of characters transmitted (not including the `\0` in the case of `sprintf`), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its `arg` parameters under control of the `format` parameter. The format is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching zero or more `arg` parameters. The behavior of the function is undefined if there are insufficient `arg` parameters for the format. If the format is exhausted while `arg` parameters remain, the extra `arg` parameters are ignored.

Each conversion specification is introduced by the character `%`. After `%`, the following appear in sequence:

1. Zero or more flag characters, which modify the meaning of the conversion specification.
2. An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded to the field width on the left (default) or right (if the left-adjustment flag has been given); see below for flag specification.
3. A precision that gives the minimum number of digits to appear for the `d`, `o`, `u`, `x`, or `X` conversions; the number of digits to appear after the decimal point for the `e`, `E`, and `f` conversions; the maximum number of significant digits for the `g` and `G` conversions; or the maximum number of characters to be printed from a string in the `s` conversion. The format of the precision is a period (`.`) followed by a decimal digit string; a null digit string is treated as zero.

4. An optional 1 specifying that a following d, o, u, x, or X conversion character applies to an arg parameter of type long. The 1 option is ignored in this implementation because type int and type long both require 32 bits.
5. A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (*) instead of a digit string. In this case, an integer arg parameter supplies the field width or precision. The arg parameter that is actually converted is not fetched until the conversion letter is seen; therefore, the arg parameters specifying field width or precision must appear immediately before the arg parameter (if any) to be converted.

These are the flag characters and their meanings:

| | |
|-------|--|
| - | The result of the conversion will be left justified within the field. |
| + | The result of a signed conversion always begins with a sign (+ or -). |
| blank | If the first character of a signed conversion is not a sign, a space will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored. |
| # | The value is to be converted to an alternate form. For c, d, s, and u conversions, the flag has no effect. For o conversion, it increases the precision to force the first digit of the result to be a zero. For x (X) conversion, a nonzero result will have 0x (0X) prefixed to it. For e, E, f, g, and G conversions, the result will always contain a decimal point, even if no digits follow the point. (Normally, a decimal point appears in the result of these conversions only if a digit follows it.) For g and G conversions, trailing zeros in the fractional part will not be removed from the result (as they normally are). |

The conversion characters and their meanings are these:

| | |
|---------------|---|
| d, o, u, x, X | The integer arg parameter is converted to signed decimal (d), unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X), respectively; the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. |
|---------------|---|

- f** The float, double, comp, or extended arg parameter is converted to decimal notation in the form "[*-*]*ddd.ddd*", where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, it is assumed to be 6; if the precision is explicitly 0, no decimal point appears. Infinities are printed in the form "[*-*]INF", and NaNs are printed in the form "[*-*]NAN(*ddd*)", where *ddd* is a code indicating why the result is not a number.
- e, E** The float, double, comp, or extended arg parameter is converted in the form "[*-*]*d.ddd±dd*", where there is one digit before the decimal point and the number of digits after it is equal to the precision. When the precision is missing, it is assumed to be 6; if the precision is 0, no decimal point appears. The E format code produces a number with E instead of e introducing the exponent. The exponent always contains at least two digits. Infinities are printed as INF and NaNs are printed in the form "[*-*]NAN(*ddd*)", where *ddd* is a code indicating why the result is not a number.
- g, G** The float, double, comp, or extended arg parameter is printed in style f or e (or in style f or E in the case of a G format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style e is used only if the exponent resulting from the conversion is less than -4 or greater than the precision. Trailing zeros are removed from the result. A decimal point appears only if it is followed by a digit.
- c** The character arg parameter is printed.
- s** The arg parameter is taken to be a string (character pointer) and characters from the string are printed until a null character (`\0`) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. If the string pointer arg parameter has the value zero, the result is undefined; a zero arg parameter yields undefined results.
- %** Print a %; no parameter is converted.

In no case does a nonexistent or small field width cause truncation of a field. If the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by `printf` and `fprintf` are printed as if `putc` had been called.

Examples

To print a date and time in the form "Sunday, July 3, 10:02", where weekday and month are pointers to null-terminated strings:

```
printf("%s, %s %d, %.2d:%.2d", weekday, month, day, hour, min);
```

To print pi to five decimal places:

```
printf("pi = %.5f", pi());
```

Note Calling `sprintf` causes other Standard I/O functions to be loaded, even though `sprintf` doesn't perform any I/O.

See also `dec2str`, `ecvt`, `num2dec`, `putc`, `scanf`, `stdio`

putc—put character or word on a stream

Synopsis

```
#include <stdio.h>

int putc(c, stream)
    char c;
    FILE *stream;
int putchar(c)
    char c;
int fputc(c, stream)
    char c;
    FILE *stream;
int putw(w, stream)
    int w;
    FILE *stream;
```

Description

Macro `putc` writes the character `c` to the output stream at the current position of the file pointer. Macro `putchar(c)` is equivalent to

```
putc(c, stdout)
```

Function `fputc` behaves like macro `putc`. Function `fputc` runs more slowly than macro `putc` but takes less space per invocation.

Function `putw` writes an `int` (that is, four bytes) to the output stream at the current position of the file pointer. This function neither assumes nor causes special alignment in the file.

For information about buffering of output files, see the `stdio` page.

Return values

On success, these functions each return the value they have written. On failure, they return the constant `EOF`. This occurs if the file stream is not open for writing or if the output file cannot be grown. Because `EOF` is a valid integer, `ferror` should be used to detect `putw` errors.

Note

Because it is implemented as a macro, `putc` treats a stream parameter with side effects incorrectly. In particular,

```
putc(c, *f++)
```

produces unexpected results. Instead use

```
fputc(c, *f++)
```

See also

`fclose`, `ferror`, `fopen`, `fread`, `getc`, `printf`, `puts`, `setbuf`, `stdio`

puts—write a string to a stream

Synopsis

```
#include <StdIO.h>

int puts(str)
    char *str;
int fputs(str, stream)
    char *str;
    FILE *stream;
```

Description

Function `puts` writes the null-terminated string pointed to by `str`, followed by a newline character, to the standard output stream `stdout`.

Function `fputs` writes the null-terminated string pointed to by `str` to the named output stream `stream`.

Neither function writes the terminating null character.

Return value

Both routines return the number of characters written, or an EOF if there is a write error.

Note

Function `puts` appends a newline character, while `fputs` does not.

See also

`ferror`, `fopen`, `fread`, `printf`, `putc`, `stdio`

qsort—quicker sort

Synopsis

```
void qsort(base, nelem, elsize, compar)
    char *base;
    unsigned int nelem, elsize;
    int (*compar)();
```

Description

Function `qsort` is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

Parameter `base` points to the element at the base of the table. Parameter `nelem` is the number of elements in the table. Parameter `elsize` is the size of an element in the table; it can be specified as `sizeof(*base)`.

Parameter `compar` is a pointer to a comparison function that you supply. Function `qsort` calls your comparison function with pointers to two elements being compared. Here is a sample declaration for your comparison function:

```
int myCompare(elem1, elem2)
    char *elem1, *elem2;
```

Your comparison function supplies the result of the comparison to `qsort` by returning one of the following integer values:

| Result | Meaning |
|--------|---|
| <0 | The first parameter is less than the second parameter. |
| 0 | The first parameter is equal to the second parameter. |
| >0 | The first parameter is greater than the second parameter. |

Note

Parameter `base`, the pointer to the base of the table, should be of type pointer-to-element and cast to `(char *)`.

rand—a simple random-number generator

Synopsis

```
int rand()
void srand(seed)
    unsigned seed; /
```

Description

Function `rand` uses a multiplicative congruential random-number generator with period 2^{32} that returns successive pseudorandom numbers in the range from 0 to $2^{15}-1$.

Function `srand` can be called at any time to reset the random-number generator to a specific seed. The generator is initially seeded with a value of 1. Identical seeds produce identical sequences of pseudorandom numbers.

See also

Random, randomx
Apple Numerics Manual

read—read from file

Synopsis

```
int read(fildes, buf, nbyte)
    int    fildes;
    char   *buf;
    unsigned nbyte;
```

Description

File descriptor `fildes` is obtained from a call to `open`, `creat`, `dup`, or `fcntl`.

Function `read` transfers up to `nbyte` bytes from the file associated with `fildes` into the buffer pointed to by `buf`.

On devices capable of seeking, `read` starts reading at the current position of the file pointer associated with `fildes`. Upon return from `read`, the file pointer is incremented by the number of bytes actually read.

Nonseeking devices always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, `read` returns the number of bytes actually read and placed in the buffer; this number may be less than `nbyte` if the file is associated with a window or if the number of bytes left in the file is less than `nbyte` bytes. A value of 0 is returned when an end of file has been reached, or -1 if a read error occurred.

Function `read` fails if `fildes` is not a valid file descriptor open for reading. [EBADF]

File descriptor 0 is opened by the MPW Shell as the standard input.

Return value

Upon successful completion, a nonnegative integer is returned indicating the number of bytes actually read. Otherwise, -1 is returned and `errno` is set to indicate the error.

See also

`creat`, `open`

scanf—convert formatted input

Synopsis

```
#include <StdIO.h>

int scanf(format [ , pointer ] ... )
    char *format;
int fscanf(stream, format [ , pointer ] ... )
    FILE *stream;
    char *format;
int sscanf(str, format [ , pointer ] ... )
    char *str, *format;
```

Description

Function `scanf` reads characters from the standard input stream `stdin`. Function `fscanf` reads characters from the named input stream `stream`. Function `sscanf` reads characters from the character string `str`. Each function converts the input according to a control string (`format`) and stores the results according to a set of pointer parameters that indicate where the converted output should be stored.

Parameter `format`, the control string, contains specifications that control the interpretation of input sequences. The format consists of characters to be matched in the input stream and/or conversion specifications that start with the character `%`. The control string may contain

- White-space characters (spaces and tabs) that cause input to be read up to the next non-white-space character, except as described below.
- A character (any except `%`) that must match the next character of the input stream. To match a `%` character in the input stream, use `%%`.
- Conversion specifications beginning with the character `%` and followed by an optional assignment suppression character `*`, an optional numeric maximum field width, an optional `l`, `m`, `n`, or `h` indicating the size of the receiving parameter, and a conversion code.

An input field is defined relative to its conversion specification. The input field ends when the first character inappropriate for conversion is encountered or when the specified field width is exhausted. After conversion, the input pointer points to the inappropriate character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding parameter, which is a pointer to a basic C type such as `int` or `float`.

Assignment can be suppressed by preceding a format character with the character `*`. *Assignment suppression* means an input field is skipped; the field is read and converted but not assigned. Therefore, pointer should be omitted when assignment of the corresponding input field is suppressed.

The *format character* dictates the interpretation of the input field. The following format characters are legal in a conversion specification, after %:

- | | |
|---------|---|
| % | A single % is expected in the input at this point; no assignment is done. |
| d | A decimal integer is expected; the corresponding parameter should be an integer pointer. |
| u | An unsigned decimal integer is expected; the corresponding parameter should be an unsigned integer pointer. |
| o | An octal integer is expected; the corresponding parameter should be an integer pointer. |
| x | A hexadecimal integer is expected; the corresponding parameter should be an integer pointer. |
| | The conversion characters d, u, o, and x may be preceded by l or h to indicate that a pointer to long or short, rather than int, is in the parameter list. The l is ignored in this implementation because int and long are both 32 bits. |
| e, f, g | <p>A floating-point number is expected; the next field is converted accordingly and stored through the corresponding parameter, which should be a pointer to a float, double, comp, or extended, depending on the size specification. The input format for floating-point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of E or e followed by an optionally signed integer. In addition, infinity is represented by the string "INF", and NaNs are represented by the string "NaN", optionally followed by parentheses that may contain a string of digits (the NaN code). Case is ignored in the infinity and NaN strings.</p> <p>The conversion characters e, f, and g may be preceded by l, m, or n to indicate that a pointer to double, comp, or extended, rather than float, is in the parameter list.</p> |
| s | A character string is expected; the corresponding parameter should be a character pointer to an array of characters large enough to accept the string; a terminating null character (\0) is added automatically. The input field is terminated by a white-space (blank or tab) character, or when the number of characters specified by the maximum field width has been read. |
| c | A character is expected; the corresponding parameter should be a character pointer. The normal skip over white space is suppressed in this case; use %1s to read the next non-white-space character. If a field width is given, the corresponding parameter should refer to a character array; the indicated number of characters is read. |

- [The left bracket introduces a *scanset* format. The input field is the maximal sequence of input characters consisting entirely of characters in the scanset. When reading the input field, string data and the normal skip over leading white space are suppressed. The corresponding pointer parameter must point to a character array large enough to hold the input field and the terminating null character (`\0`), which will be added automatically. The left bracket is followed by a set of characters (the scanset) and a terminating right bracket.
 - ^ When it appears as the first character in the scanset, the circumflex serves as a complement operator and redefines the scanset as the set of all characters not contained in the remainder of the scanset string.
 -] The right bracket ends the scanset. To include the right bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset. Otherwise, it will be interpreted syntactically as the closing bracket.
- A range of characters may be represented by the construct *first-last*; thus the scanset `[0123456789]` may be expressed `[0-9]`. To use this convention, *first* must be less than or equal to *last* in the ASCII collating sequence. Otherwise, the minus (`-`) will stand for itself in the scanset. The minus will also stand for itself whenever it is the first or the last character in the scanset.

Conversion terminates at EOF, at the end of the control string, or when an input character doesn't match the control string. In the last case, the unmatched character is left unread in the input stream.

Examples

Example 1

The call

```
int i;
float x;
char name[50];
scanf("%d%f%s", &i, &x, name);

with input

25 .54.32E-1 hartwell
```

will assign the value 25 to `i` and the value 5.432 to `x`; `name` will contain `"hartwell\0"`.

Example 2

The call

```
int i;
extended x;
char name[50];
scanf("%2d%nf%d %[0-9]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

will assign 56 to *i* and 789.0 to *x*, skip 0123, and place the string "56\0" in *name*. The next call to `getchar` will return "a".

Example 3

The call

```
int i;
scanf("answer1=%d", &i);
```

with input

```
answer1=51 answer2=45
```

will assign the value 51 to *i* because "answer1" is matched explicitly in the input stream; the input pointer will be left at the space before "answer2".

Return value Functions `scanf`, `fscanf`, and `sscanf` return the number of successfully matched and assigned input items; this number can be 0 when an early mismatch between an input character and the control string occurs. If the input ends before the first mismatch or conversion, EOF is returned.

These functions return EOF on end of input and a short count for missing or illegal data items.

Note Trailing white space is left unread unless matched in the control string. The success of literal matches and suppressed assignments is not directly determinable.

Warning The pointer parameters in these functions must be addresses—for example, `&i`. Be sure not to pass *i* rather than its address.

See also `atof`, `dec2num`, `getc`, `printf`, `stdio`, `str2dec`, `strtol`
Apple Numerics Manual

setbuf—assign buffering to a stream

Synopsis

```
#include <StdIO.h>

void setbuf(stream, buf)
    FILE *stream;
    char *buf;
int setvbuf(stream, buf, type, size)
    FILE *stream;
    char *buf;
    int type;
    int size;
```

Description

A buffer is normally allocated by the Standard C Library at the time of the first `getc` or `putc` on a file. If you prefer to provide your own buffer, you can call `setbuf` or `setvbuf` after a stream has been associated with an open file but before it is read or written. Functions `setbuf` and `setvbuf` let you provide your own buffering for a file stream. Function `setvbuf` is a more flexible extension of `setbuf`.

Function `setbuf` causes the character array pointed to by `buf` to be used instead of an automatically allocated buffer. `BUFSIZ`, a constant defined in the `<StdIO.h>` header file, lets you specify the size of the `buf` array as

```
char buf[BUFSIZ];
```

If `buf` is `NULL`, input/output is unbuffered.

Function `setvbuf` lets you specify two parameters in addition to those required by `setbuf`: `size` and `type`. Parameter `size` specifies the size in bytes of the array to be used; the standard I/O functions work most efficiently when `size` is a multiple of `BUFSIZ`. If buffer pointer `buf` is `NULL`, a buffer of `size` bytes is allocated from the system. If `size` is not 0, `size` is assigned to the `FILE` variable's `size` parameter; if `buf` is not `NULL`, `buf` is assigned to the `FILE` variable's `buffer-pointer` parameter. The value of `type` determines how `stream` is buffered by `setvbuf`:

| Value of type | Description |
|---------------------|--|
| <code>_IOFBF</code> | Causes input/output to be file buffered. |
| <code>_IOLBF</code> | Causes output to be line buffered. The buffer is flushed when a newline character is written or when the buffer is full. |
| <code>_IONBF</code> | Causes input/output to be unbuffered. Parameters <code>buf</code> and <code>size</code> are ignored. |

The following function calls are equivalent when `buf` is not `NULL`:

```
setbuf(stream, buf);
setvbuf(stream, buf, _IOFBF, BUFSIZ);
```

The following function calls are equivalent when `buf` is `NULL`:

```
setbuf(stream, NULL;  
setvbuf(stream, NULL, _IONBF, BUFSIZ);
```

Diagnostics Function `setvbuf` returns nonzero if an invalid value is given for `type`.

Note The buffer must have a lifetime at least as great as the open stream. Be sure to close the stream before the buffer is deallocated. If you allocate buffer space as an automatic variable in a code block, be sure to close the stream in the same block.
If `buf` is `NULL` and the system cannot allocate `size` bytes, a smaller buffer will be allocated.

See also `fopen`, `getc`, `malloc`, `putc`, `stdio`

setjmp—nonlocal transfer of control

Synopsis

```
#include <SetJmp.h>

int setjmp(env)
    jmp_buf env;
void longjmp(env, val)
    jmp_buf env;
    int val;
```

Description

These functions let you escape from an error or interrupt encountered in a low-level subroutine of your program.

Function `setjmp` saves its stack environment in `env` for later use by `longjmp`. It returns the value 0.

Function `longjmp` restores the environment saved by the last call of `setjmp` with the corresponding `env` environment. After a call to `longjmp`, the program continues as if the preceding call to `setjmp` had returned the value `val`.

Function `longjmp` cannot cause `setjmp` to return the value 0. If `longjmp` is invoked with a second parameter of 0, `setjmp` returns 1. Data values will be those in effect at the time `longjmp` was called, except for register variables (see “Warning”).

Warning

If `longjmp` is called without a previous call to `setjmp` or if the function that contained the `setjmp` has already returned, results are unpredictable.

After a `longjmp`, variables that happen to be assigned to registers are restored to their values before the call to `setjmp`, instead of those in effect at the time `longjmp` was called. To avoid this, declare “important” variables as static. (This will prohibit their use as register variables.)

See also

`signal`

signal—signal handling

Synopsis

```
#include <Signal.h>

typedef unsigned short SignalMap;
typedef int SignalHandler;
SignalHandler *sigset(sigMap, newHandler)
    SignalMap    sigMap;
    SignalHandler *newHandler;
void _sig_dfl(sigNo, sigState, sigEnabled)
    SignalMap    sigNo;
    SignalMap    sigState;
    SignalMap    sigEnabled;
SignalMap sighold(sigMap)
    SignalMap    sigMap;
void sigrelease(sigMap, prevEnabled)
    SignalMap    sigMap;
    SignalMap    prevEnabled;
void sigpause(sigMap)
    SignalMap    sigMap;
```

Description

C programs that handle software interrupts—known as **signals**—should use these procedures, which support signal handling under MPW. A signal is similar to a hardware interrupt in that its invocation can cause program control to be temporarily diverted from its normal execution sequence; the difference is that the events that raise a signal reflect a change in program state rather than hardware state. Examples of signal events are stack overflow, heap overflow, software floating-point errors, and Command-period interrupts.

Signal handling is available only for tools that run under the MPW Shell; it is not available for applications that run under the Macintosh Finder.

Currently, the only software interrupt provided is Command-period, which is represented by the value SIGINT. As additional software interrupts are provided, new values will be added to represent them; the signal-handling procedures will then accept these new signals.

Signals can be caught, held and released, or ignored. The default action of any signal raised is to close all open files, execute any exit procedures installed with `onexit`, and terminate the program. No signal-handling calls are required to execute a normal termination on receipt of a signal. If a program requires special handling of a signal or chooses to ignore it, `sigset` lets you replace the default procedure with a user procedure. You can also temporarily "hold" (that is, suspend) action on a signal by calling `sighold`. You may want to do this before entering a critical section of code. The signal can then be restored by calling the procedure `sigrelease`, whereupon its signal-handling procedure will take effect if the signal was raised since the preceding call to `sighold`. Your program may also wait until one or more signals are raised by calling the `sigpause` procedure.

A signal is represented by a bit in the integer `SignalMap`, which identifies one or more signals to the signal-handling procedures. You can refer to several signals at once by adding or OR-ing their bits together. You can refer to all signals at once by using the value `SIGALLSIGS`.

The `sigset` function: Function `sigset` replaces the current signal handler (the procedure to be executed upon receipt of the signals specified in `sigMap`) with a user-supplied signal handler. The default signal handler may be set or restored by specifying `SIG_DFL` as the current signal handler. The signals may be ignored entirely by specifying `SIG_IGN` as the current signal handler.

Function `sigset` returns the previous `SignalHandler` pointer. If this pointer must be restored in another part of the program, save the return value and restore it with another call to `sigset`. Multiple signals may be set with one call to `sigset` by OR-ing signal values together in `sigMap`, but in this case `sigset` cannot, of course, return all previous values and its return value is meaningless. To correctly save multiple previous signal handlers, call `sigset` separately for each signal.

The `sig_dfl` function: This is the default procedure `SIG_DFL`; it is not intended for use by the program directly. It is documented here as an example of a user-supplied signal handler that uses standard C calling conventions.

The first parameter, `sigNo`, is the signal that is being raised. Although it is declared as a `SignalMap`, its value contains at most one signal bit; it can therefore be compared for equality against a signal name, for example, `SIGINT`. The same signal handler may trap several signals with common code and then inspect `sigNo` if special handling of particular signals is required.

The parameters `sigState` and `sigEnabled` provide runtime information about current active signals. Bit map `sigState` describes all raised signals, including signals held by calls to `sighold`. Bit map `sigEnabled` describes all signals currently enabled. By default, all signals are enabled, but they may be disabled by holding them.

Upon entry to a user-supplied signal handler, all signals are temporarily suspended; therefore, the handler is not required to lock out recursive or nested calls to signal handlers. The signal state is restored upon normal return from the signal handler.

Signals cannot be raised while executing in ROM or in the MPW Shell. If a signal event occurs while executing outside the tool, the signal state is set and the signal handler is executed as soon as program control returns to the tool. Because a signal can interrupt the tool at any point, there is no protection against heap corruption if a signal handler executes calls that modify the state of the heap. Because most buffered I/O potentially modifies the heap, `printf` and similar calls are not recommended in signal handlers unless they call `exit` to avoid returning to the application program. Even then, the caller must be careful of interaction between `exit` and `onexit` procedures.

The `sighold` function: The `sighold` function, like `sigrelease`, permits temporary suspension and restoration of signals. Before a program enters a critical section of code, it should call `sighold` with a signal map of signals to suspend or with the value `SIGALLSIGS`, which represents all signals. Function `sighold` returns a `SignalMap` representing the list of signals already being held; this value should be saved for use as the `prevEnabled` parameter in the subsequent call to `sigrelease`. If the signal event (such as `Command-period`) occurs after a call to `sighold` is made, the event is recorded in the signal state but the signal handler is not executed.

The `sigrelease` function: Function `sigrelease` lets you reenable signals that were held by a previous call to `sighold` by specifying their corresponding bits in `sigMap`. Signals that were already on hold when you called `sighold` should be specified to `sigrelease` in the `prevEnabled` parameter to permit correct handling of nested calls to `sighold`. If any of the signal events occurred while they were held, their signal-handling routines will take effect immediately after the return from `sigrelease`. Signal events do not stack; multiple occurrences of signal events that are being held do not yield multiple invocations of the signal handler when the signal is released.

The `sigpause` function: A call to `sigpause` suspends program activity until a signal event is recorded for any signal not currently held. It is intended for signal synchronization, though in the current implementation its application is limited; it is included here in order to provide a complete signal environment model.

sinh—hyperbolic functions

Synopsis

```
#include <Math.h>

extended sinh(x)
    extended x;
extended cosh(x)
    extended x;
extended tanh(x)
    extended x;
```

Description

Functions `sinh`, `cosh`, and `tanh` return, respectively, the hyperbolic sine, cosine, and tangent of their parameter.

Diagnostics

Functions `sinh`, `cosh`, and `tanh` honor the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by SANE.

See also

Apple Numerics Manual

stdio—standard buffered input/output package

Synopsis

```
#include <StdIO.h>

FILE *stdin, *stdout, *stderr;
```

Description

The Standard I/O package constitutes an efficient user-level I/O buffering scheme. The inline macros `getc` and `putc` handle characters quickly. Macros `getchar` and `putchar`, and the higher-level routines `fgetc`, `fgets`, `fprintf`, `fputc`, `fputs`, `fread`, `fscanf`, `fwrite`, `gets`, `getw`, `printf`, `puts`, `putw`, and `scanf`, all use `getc` and `putc`; calls to these macros and functions can be freely intermixed.

The constants and the following functions are implemented as macros: `getc`, `getchar`, `putc`, `putchar`, `feof`, `ferror`, `clearerr`, and `fileno`. Redclaration of these names should be avoided.

Any program that uses the Standard I/O package must include the `<StdIO.h>` header file of macro definitions. The functions, macros, and constants used in the Standard I/O package are declared in the header file and need no further declaration.

A *stream* is a file with associated buffering and is declared to be a pointer to a `FILE` variable. Functions `fopen`, `freopen`, and `fdopen` return this pointer. The information in the `FILE` variable includes

- ☐ the file access—read or write
- ☐ the file descriptor as returned by `open`, `creat`, `dup`, or `fcntl`
- ☐ the buffer size and location
- ☐ the buffer style (unbuffered, line-buffered, or file-buffered)

Standard I/O buffering: Output streams, with the exception of the standard error stream `stderr`, are by default file buffered if the output refers to a file. File `stderr` is by default line buffered. When an output stream is *unbuffered*, it is queued for writing on the destination file or window as soon as written; when it is *file buffered*, many characters are saved up and written as a block; when it is *line buffered*, each line of output is queued for writing as soon as the line is completed (that is, as soon as a newline character is written). Function `setvbuf` may be used to change the stream's buffering strategy.

Normally, there are three open streams with constant pointers declared in the `<StdIO.h>` header file and associated with the standard open files:

| FILE variable | Fildes | Description | Buffer style |
|---------------------|--------|----------------------|---------------|
| <code>stdin</code> | 0 | standard input file | file buffered |
| <code>stdout</code> | 1 | standard output file | file buffered |
| <code>stderr</code> | 2 | standard error file | line buffered |

Buffer initialization: The `FILE` variable returned by `fopen`, `freopen`, or `fdopen` has an initial buffer size of 0 and a `NULL` buffer pointer. The buffer size is set and the buffer allocated by a call to `setbuf`, `setvbuf`, or the first I/O operation on the stream, whichever comes first. Buffer initialization is done using the following algorithm:

1. If `_IONBF` (no buffering) was set by a call to `setvbuf`, initialization steps 2 and 3 are skipped. The buffer size remains 0 and the buffer pointer remains `NULL`.
2. Checks the access-mode word for `_IOLBF` (line buffering). This bit is usually set only in the predefined file `stderr`, but a call to `setvbuf` can set it for any file. If line buffering is set, the buffer size is set to `LBUFSIZ` (100). If line buffering is not set, `ioctl` is called with an `FIOBUFSIZE` request and the buffer size is set to the returned value or to `BUFSIZ` (1024) if no value is returned.
3. If the buffer pointer is `NULL`, a request is made for a buffer whose size was determined in step 2; the buffer pointer is set to point to the newly allocated buffer. If the requested size cannot be allocated, attempts are made to allocate `BUFSIZ` or `LBUFSIZ` if these are smaller than the requested size. If all requests fail, the buffer pointer remains `NULL` and the `_IONBF` (no buffering) bit is set.
4. Function `ioctl` is called with an `FIOINTERACTIVE` request; if it returns `true`, the `_IOSYNC` bit is set in the access-mode word. This is done for all `FILE` variables, regardless of their buffering style and size. (The `_IOSYNC` bit is described in the following section.)

The `setvbuf` function lets you specify values for buffer size, buffer pointer, and access mode word other than the default values of 0, `NULL`, and 0, respectively. The `setvbuf` function must be called before the first I/O operation occurs, so that the buffer initialization procedure described above receives the values you specify instead of the default values.

Buffered I/O: On each write request, the bytes are transferred to the buffer and an internal counter is set to account for the number of bytes in the buffer. If `_IOLBF` is set and a newline character is encountered while transferring bytes to the buffer, the buffer is flushed (written immediately) and the transfer continues at the beginning of the buffer. This continues until the write-request count is satisfied or a write error occurs.

On each read request, the `_IOSYNC` bit in the access-mode word is checked. If `_IOSYNC` is on, all current `FILE` variables that have `_IOSYNC` on and are open for writing are flushed. In other words, a read from an interactive `FILE` variable flushes all interactive output files before reading. This ensures that any prompts, I/O in a window, or other visual feedback is displayed before the read is initiated. Then if the internal counter is 0, an entire buffer is read into memory if possible. (For the console device, less than a buffer's worth is likely to be read.) The bytes required to satisfy the read request are transferred, going back to the device for more if necessary, and an internal pointer is advanced if any bytes remain unread.

When the Standard I/O package is used, Standard I/O cleanup is performed just before termination of the application. Any normal return including a call to `exit` causes Standard I/O cleanup, which consists of a call to `fclose` for every open `FILE` stream.

- Note** Do not use a file descriptor (0, 1, or 2) where a `FILE` variable (`stdin`, `stdout`, or `stderr`) is required.
- File `<StdIO.h>` includes definitions other than those described above, but their use is not recommended.
- Invalid stream pointers cause serious errors, possibly including program termination. Individual function descriptions describe the possible error conditions.
- Diagnostics** An integer constant `EOF` (-1) is returned upon end of file or error by most integer functions that deal with streams. See the descriptions of the individual functions for details.
- See Also** `open`, `close`, `lseek`, `read`, `write`, `fclose`, `ferror`, `fopen`, `fread`, `fseek`, `getc`, `gets`, `printf`, `putc`, `puts`, `scanf`, `setbuf`, `ungetc`

string—string operations

Synopsis

```
char *strcat(destStr, srcStr)
    char *destStr, *srcStr;
char *strncat(destStr, srcStr, n)
    char *destStr, *srcStr;
    int n;
int strcmp(str1, str2)
    char *str1, *str2;
int strncmp(str1, str2, n)
    char *str1, *str2;
    int n;
char *strcpy(destStr, srcStr)
    char *destStr, *srcStr;
char *strncpy(destStr, srcStr, n)
    char *destStr, *srcStr;
    int n;
int strlen(str)
    char *str;
char *strchr(str, c)
    char *str, c;
char *strrchr(str, c)
    char *str, c;
char *strpbrk(srcStr, findChars)
    char *srcStr, *findChars;
int strspn(srcStr, spanChars)
    char *srcStr, *spanChars;
int strcspn(srcStr, skipChars)
    char *srcStr, *skipChars;
char *strtok(destStr, tokenStr)
    char *destStr, *tokenStr;
```

Description

The string parameters (*srcStr*, *destStr*, and so forth) and *s* point to arrays of characters terminated by a null character. The functions *strcat*, *strncat*, *strcpy*, and *strncpy* all alter *destStr*. These functions do not check for overflow of the array pointed to by *destStr*.

Function *strcat* appends a copy of string *srcStr* to the end of string *destStr*. Function *strncat* appends at most *n* characters. Each function returns a pointer to the null-terminated result.

Function *strcmp* performs a comparison of its parameters according to the ASCII collating sequence and returns an integer less than, equal to, or greater than 0 when *str1* is less than, equal to, or greater than *str2*, respectively. Function *strncmp* makes the same comparison but looks at a maximum of *n* characters.

Function `strcpy` copies string `srcStr` to string `destStr`, stopping after the null character has been copied. Function `strncpy` copies exactly `n` characters, truncating `srcStr` or adding null characters to `destStr` if necessary. The result is not terminated with a null if the length of `srcStr` is `n` or more. Each function returns `destStr`.

Function `strlen` returns the number of characters in `str`, not including the terminating null character.

Functions `strchr` and `strrchr` both return a pointer to the first and last occurrence, respectively, of character `c` in string `str`; they return a null pointer if `c` does not occur in the string. The null character terminating a string is considered to be part of the string. In previous versions of the Standard C Library, `strchr` was known as `index` and `strrchr` was known as `rindex`.

Function `strpbrk` returns a pointer to the first occurrence in string `srcStr` of any character from string `findChars`, or a null pointer if no character from `findChars` exists in `srcStr`.

Function `strspn` returns the length of the initial segment of string `srcStr` that consists entirely of characters from string `spanChars`.

Function `strcspn` returns the length of the initial segment of string `srcStr` that consists entirely of characters not from string `skipChars`.

Function `strtok` considers the string `destStr` as a sequence of zero or more text tokens separated by spans of one or more characters from the separator string `tokenStr`. The first call (with pointer `destStr` specified) returns a pointer to the first character of the first token and writes a null character into `destStr` immediately following the returned token. The function keeps track of its position in the string between calls. Subsequent calls for the same string must be made with a null pointer as the first parameter. The separator string `tokenStr` may be different from call to call. When no token remains in `destStr`, a null pointer is returned.

Warning

Overlapping moves yield unexpected results.

Functions `strcmp` and `strncmp` use signed arithmetic when comparing their parameters. The sign of the result will be incorrect for characters with values greater than `0x7F` in the Macintosh extended character set.

See also

`BlockMove`, `EqualString`, `memory`

strtol—convert a string to a long

Synopsis

```
long strtol(str, ptr, base)
    char    *str;
    char    **ptr;
    int     base;
```

Description

Function `strtol` returns a long containing the value represented by the character string `str`. The string is scanned up to the first character inconsistent with the base (decimal, hexadecimal, or octal). Leading white-space characters are ignored.

If the value of `ptr` is not `NULL`, a pointer to the character terminating the scan is returned in `*ptr`. If no integer can be formed, `*ptr` is set to `str` and 0 is returned.

If `base` is 0, the base is determined from the string. If the first character after an optional leading sign is not 0, decimal conversion is done; if the 0 is followed by `x` or `X`, hexadecimal conversion is done; otherwise octal conversion is done.

The function call `atol(str)` is equivalent to

```
strtol(str, (char **)NULL, 10)
```

The function call `atoi(str)` is equivalent to

```
(int) strtol(str, (char **)NULL, 10)
```

Note

Overflow conditions are ignored.

Apple base conventions (`$` for hexadecimal, `%` for binary) are not supported.

See Also

`atof`, `atoi`, `scanf`

trig—trigonometric functions

Synopsis

```
#include <Math.h>
extended sin(x)
    extended x;
extended cos(x)
    extended x;
extended tan(x)
    extended x;
extended asin(x)
    extended x;
extended acos(x)
    extended x;
extended atan(x)
    extended x;
extended atan2(y, x)
    extended y, x;
```

Description

Functions `sin`, `cos`, and `tan` return, respectively, the sine, cosine, and tangent of their argument, which is in radians.

Function `asin` returns the arcsine of `x`, in the range $-\pi/2$ to $\pi/2$.

Function `acos` returns the arccosine of `x`, in the range 0 to π .

Function `atan` returns the arctangent of `x`, in the range $-\pi/2$ to $\pi/2$.

Function `atan2` returns the arctangent of `y/x`, in the range $-\pi$ to π , using the signs of both arguments to determine the quadrant of the return value.

For special cases, these functions return a NaN or infinity as appropriate.

Diagnostics

These functions honor the floating-point exception flags—invalid operation, underflow, overflow, divide by zero, and inexact—as prescribed by SANE.

Note

Functions `sin`, `cos`, and `tan` have periods based on the nearest extended-precision representation of mathematical π . Hence these functions diverge from their mathematical counterparts as their argument becomes far from zero.

See also

Apple Numerics Manual

ungetc—push a character back into the input stream

Synopsis

```
#include <StdIO.h>

int ungetc(c, stream)
    char    c;
    FILE    *stream;
```

Description

Function `ungetc` inserts the character `c` into the buffer associated with an input stream. The stream must be file buffered or line buffered; it cannot be unbuffered. The inserted character, `c`, will be returned by the next `getc` call on that stream. Function `ungetc` returns `c` and leaves the file stream unchanged.

Only one character of pushback is allowed, provided something has been read from the stream and the stream is not unbuffered.

If `c` equals EOF, `ungetc` does nothing to the buffer and returns EOF. In other words, you cannot use `ungetc` to force end of file the next time the file is read.

Function `fseek` undoes the effect of `ungetc`.

Diagnostics

For `ungetc` to perform correctly, a read must have been performed before the call to the `ungetc` function. Function `ungetc` returns EOF if it can't insert the character.

Note

Function `ungetc` does not work on unbuffered streams.

See also

`fseek`, `getc`, `setbuf`, `stdio`

unlink—delete a named file

Synopsis

```
int unlink(fileName)
    char      *fileName;
```

Description

Function `unlink` deletes the named file. The function fails if the named file is open. A call to `unlink` is equivalent to

```
faccess(fileName, F_DELETE)
```

Diagnostics

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

See also

`faccess`

write—write on a file

Synopsis

```
int write(fildes, buf, nbyte)
    int      fildes;
    char     *buf;
    unsigned  nbyte;
```

Description

File descriptor `fildes` is obtained from an `open`, `creat`, `dup`, or `fcntl` call.

Function `write` attempts to write `nbyte` bytes from the buffer pointed to by `buf` to the file associated with the `fildes`. Internal limitations may cause `write` to write fewer bytes than requested; the number of bytes actually written is indicated by the return value. Several calls to `write` may therefore be necessary to write out the contents of `buf`.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from `write`, the file pointer is incremented by the number of bytes actually written.

On nonseeking devices, writing starts at the current position. The value of a file pointer associated with such a device is undefined.

If the `O_APPEND` file status flag set in `open` is on, the file pointer is set to end of file before each write.

The file pointer remains unchanged and `write` fails if `fildes` is not a valid file descriptor open for writing. [EBADF]

If you try to write more bytes than there is room for on the device, `write` writes as many bytes as possible. For example, if `nbyte` is 512 and there is room for 20 bytes more on the device, `write` writes 20 bytes and returns a value of 20. The next attempt to write a nonzero number of bytes will return an error. [ENOSPC]

File descriptor 1 is standard output; file descriptor 2 is standard error.

Return value

Upon successful completion, the number of bytes actually written is returned. Otherwise, `-1` is returned and `errno` is set to indicate the error.

See also

`creat`, `lseek`, `open`



Chapter 4



The Macintosh Interface Libraries

About the Macintosh Interface Libraries

This chapter contains the C definitions of the constants, types, and functions defined in *Inside Macintosh*. The information here is the C equivalent of the Pascal definitions in the Summary sections at the end of each chapter of *Inside Macintosh*. Complete documentation for each of the constants, types, and functions defined here is found in the corresponding section of *Inside Macintosh*.

- ❖ *Note:* If you have a Macintosh Plus or a Macintosh 512K enhanced—with 128K ROM—you have access to all of the Macintosh Interface Libraries. If you have a Macintosh with 64K ROM, you have access only to the libraries documented in *Inside Macintosh*, Volumes 1-3; you don't have access to those documented in *Inside Macintosh*, Volume 4. Parts of the FixMath library are available in a RAM library for 64K ROM users. Graf3D is available in a library for all users.

After an introductory description of the interface, this chapter is arranged alphabetically by library name. The libraries are documented in the manner described in the section "About the Standard C Library" in Chapter 3. All of the identifiers in the Macintosh Interface Libraries are listed in Appendix C, "The Library Index."

Header files

Include the ".h" header files in C programs to declare the defines, types, and functions provided by these libraries. Each library definition lists the include directives necessary for use of that library. Functions whose declarations can be inferred from calls (that is, integer functions) have been omitted from the header files, to improve the Compiler's efficiency.

How the interface is implemented

Many Macintosh Interface Libraries routines are declared as external Pascal routines with trap numbers and are trapped to directly by compiled code. You cannot take the addresses of these functions. Other routines are declared to be C routines and are called through interface code.

The interface code is contained in files {CLibraries}CInterface.o and {Libraries}Interface. Link these files with the C program and other libraries. Not all functions require interface code. The Linker includes interface code for only those routines that are called.

Parameter types

The C interfaces expect small structures, like `Points`, to be passed by address. String parameters are null-terminated C strings unless otherwise indicated. `ResTypes` and `OSTypes` can be expressed as character literals; for example, `'MENU'`. All `VAR` parameters in `extern pascal` declarations must be passed explicitly by address.

All structures are passed to the ROM interface procedures by means of a pointer. The ROM actually expects small structures to be passed by value: that is, if the structure is four bytes or less in size, Pascal calling conventions dictate that the structure should itself be pushed on the stack. When this convention disagrees with what the ROM expects, assembly-language "glue" procedures provide the proper interface.

Passing string parameters

In general, C programmers use strings like

```
"hello, world\n"
```

which is an array of characters whose last element is the null byte (`'\0'`).

The Macintosh ROM, however, expects Pascal strings, which have an initial byte of count with the string following (and no null byte at the end).

Because both strings have an extra byte of information (either a count at the beginning or a null byte at the end), it is possible to transform a string *in place* from a Pascal string into a C string and vice versa. The routines `c2pstr()` and `p2cstr()` in the library `CInterface.o` perform these conversions (see the Strings page in this chapter).

The interface routines in `CInterface.o` do these conversions for you automatically. Whenever you call a ROM interface routine and one of the parameters is a string (that is, a pointer to `char`, of type `Str255` in *Inside Macintosh*), you should pass a C string. Then the interface routine performs the following actions:

1. converts the input strings from C strings into Pascal strings
2. fixes up the stack so the parameters conform to Pascal calling conventions
3. calls the ROM
4. converts the output strings from Pascal strings back into C strings

Therefore you always pass C strings to the ROM interface procedures and receive C strings in return.

Note that the conversions happen only when the string is as *parameter* to the interface routine. If the string is a field of a structure passed to the ROM, then *no* conversion is performed.

Any exceptions to this rule will be noted in the Warning section of the appropriate manual page in this chapter.

Finally, if you have a Pascal string and want to convert it into a C string (for example, before you call a ROM interface procedure), you can do so with the routine `p2cstr()`. The routine `c2pstr()` converts a C string into a Pascal string. Both conversions are done in place.

❖ *Note:* If you want to create your own Pascal string, simply preface your string with a byte count, in this way:

```
char *pascalstring = "\005hello";
```

This puts an extra null byte at the end, but has the following properties:

`&pascalstring[0]` is a pointer to a Pascal string.

`&pascalstring[1]` is a pointer to a C string.

Of course, in an application most strings come from resources rather than constants in your program. This allows your application to be altered with the resource editor (for internationalization or customization).

Correspondences between Pascal variant records and C structs

Some of the variant records in *Inside Macintosh* are implemented in the MPW C header files by means of multiple distinct struct declarations. (The reader might have expected unions.) Table 4-1 is a list of variant record types, with references to *Inside Macintosh* and to the MPW C manual page (in this chapter) and corresponding C struct name.

Table 4-1
Correspondences between variant records and structs

| <i>Inside Macintosh</i> | | <i>MPW C Reference</i> | |
|-------------------------|--------------------|------------------------|-------------|
| variant record | Chapter (volume) | struct | Manual page |
| ABusRecord | AppleTalk [2] | ATLAPRec | AppleTalk |
| ABusRecord | AppleTalk [2] | ATDDPRec | AppleTalk |
| ABusRecord | AppleTalk [2] | ATNBPRC | AppleTalk |
| ABusRecord | AppleTalk [2] | ATDDPRec | AppleTalk |
| ParamBlockRec | File Manager [2] | CntrlParam | Devices |
| ParamBlockRec | File Manager [2] | ParamBlockRec | Files |
| QElem | O. S. Utilities[2] | QElem | Osutils |
| DrvSts | Disk Driver | DrvSts | Disks |
| DrvSts | Disk Driver | DrvSts2 | Disks |
| Point | Quickdraw [1] | Point | Types |
| Rect | Quickdraw [1] | Rect | Types |

InterfaceC interface to the Macintosh libraries

Synopsis

```
#include <Types.h>      /* common defines and types */
#include <Resources.h>   /* Resource Manager */
#include <QuickDraw.h>  /* QuickDraw */
#include <Windows.h>    /* Window Manager */
#include <OSUtils.h>    /* Operating System Utilities */
#include <AppleTalk.h>  /* AppleTalk Manager */
#include <Controls.h>   /* Control Manager */
#include <Desk.h>       /* Desk Manager */
#include <Devices.h>    /* Device Manager */
#include <Dialogs.h>    /* Dialog Manager */
#include <Disks.h>      /* Disk Driver */
#include <Errors.h>     /* System Error Handler */
#include <Events.h>     /* Event Manager */
#include <Files.h>      /* File Manager */
#include <FixMath.h>    /* fixed-point arithmetic */
#include <Fonts.h>      /* Font Manager */
#include <Graf3D.h>     /* Graf3D */
#include <Lists.h>      /* List Manager */
#include <Memory.h>     /* Memory Manager */
#include <Menus.h>      /* Menu Manager */
#include <OSEvents.h>   /* OS Event Manager */
#include <Packages.h>   /* packages */
#include <Printing.h>   /* Printing Manager */
#include <Retrace.h>    /* Vertical Retrace Manager */
#include <SANE.h>       /* SANE Numerics */
#include <Scrap.h>      /* Scrap Manager */
#include <SCSI.h>       /* SCSI Manager */
#include <SegLoad.h>    /* Segment Loader */
#include <Serial.h>     /* Serial Drivers */
#include <Sound.h>      /* Sound Driver */
#include <Strings.h>    /* string conversions */
#include <TextEdit.h>   /* TextEdit */
#include <Time.h>       /* Time Manager Package */
#include <ToolUtils.h>  /* Toolbox Utilities */
```

Description

The C Interface provides C programs with access to all of the libraries defined in *Inside Macintosh*. Constants, types, and library routines are provided. The list of libraries appears in the Synopsis.

Note

List the first five `#include` directives in the order in which the libraries are listed above. The order of the other `#include` directives is irrelevant.

AppleTalk—AppleTalk Manager

Synopsis

```
#include <Types.h>
#include <AppleTalk.h>
#define lapSize          20 /* ABusRecord size for ALAP */
#define ddpSize          26 /* ABusRecord size for DDP */
#define nbpSize          26 /* ABusRecord size for NBP */
#define atpSize          56 /* ABusRecord size for ATP */
#define nbpBuffOvr      (-1024)
#define nbpNoConfirm    (-1025)
#define nbpConfDiff     (-1026)
#define nbpDuplicate    (-1027)
#define nbpNotFound     (-1028)
#define reqFailed       (-1096)
#define tooManyReqs     (-1097)
#define tooManySktS     (-1098)
#define badATPSkt       (-1099)
#define badBuffNum      (-1100)
#define cbNotFound      (-1102)
#define noSendResp      (-1103)
#define noDataArea      (-1104)
#define reqAborted      (-1105)
#define atpBadRsp       (-3107)
#define recNotFnd       (-3108)

typedef enum {
    tLAPRead,tLAPWrite,tDDPRead,tDDPWrite,tNBPLookup,
    tNBPCConfirm,tNBPCRegister,tATPSndRequest,tATPGetRequest,
    tATPSdRsp,tATPAddrRsp,tATPRequest,tATPResponse
} ABCallType; /* type of call */
typedef enum { lapProto,ddpProto,nbpProto,atpProto } ABProtoType;
typedef struct LAPAdrBlock {
    unsigned char  dstNodeID,srcNodeID,lapProtType;
} LAPAdrBlock;

typedef struct AddrBlock {
    short          aNet;
    unsigned char  aNode,aSocket;
} AddrBlock;

typedef struct EntityName {
    String(32)      objStr,typeStr,zoneStr;
} EntityName, *EntityPtr;

typedef struct RetransType {
    unsigned char  retransInterval,retransCount;
} RetransType;
typedef char BitMapType;
```

```

typedef struct BDSElement {
    short      buffSize;
    Ptr        buffPtr;
    short      dataSize;
    long       userBytes;
} BDSElement, BDSType[8];
typedef BDSType *BDSPtr;

typedef struct ATLAPRec {
    ABCallType  abOpcode; /* type of call */
    short      abResult;   /* result code */
    long       abUserReference; /* for your use */
    LAPAdrBlock lapAddress;
    short      lapReqCount;
    short      lapActCount;
    Ptr        lapDataPtr;
} ATLAPRec, *ATLAPRecPtr, **ATLAPRecHandle;

typedef struct ATDDPRec {
    ABCallType  abOpcode;
    short      abResult;
    long       abUserReference;
    short      ddpType;
    short      ddpSocket;
    AddrBlock  ddpAddress;
    short      ddpReqCount;
    short      ddpActCount;
    Ptr        ddpDataPtr;
    short      ddpNodeID;
} ATDDPRec, *ATDDPRecPtr, **ATDDPRecHandle;

typedef struct ATNBPRC {
    ABCallType  abOpcode;
    short      abResult;
    long       abUserReference;
    EntityPtr  nbpEntityPtr;
    Ptr        nbpBufPtr;
    short      nbpBufSize;
    short      nbpDataField;
    AddrBlock  nbpAddress;
    RetransType nbpRetransmitInfo;
} ATNBPRC, *ATNBPRCPtr, **ATNBPRCHandle;

typedef struct ATATPRec {
    ABCallType  abOpcode;
    short      abResult;
    long       abUserReference;
    short      atpSocket;
    AddrBlock  atpAddress;
    short      atpReqCount;
    Ptr        atpDataPtr;
    BDSPtr     atpRspBDSPtr;
    BitMapType atpBitMap;
}

```

```

    short      atpTransID;
    short      atpActCount;
    long       atpUserData;
    Boolean     atpXO;
    Boolean     atpEOM;
    short      atpTimeOut;
    short      atpRetries;
    short      atpNumBufs;
    short      atpNumRsp;
    short      atpBDSSize;
    long       atpRspUData;
    Ptr        atpRspBuf;
    short      atpRspSize
} ATATPRec, *ATATPRecPtr, **ATATPRecHandle;

/* Opening and Closing AppleTalk */

pascal short MPPOpen();
pascal short MPPClose();

/* AppleTalk Link Access Protocol */

pascal short LAPOpenProtocol(theLAPType,protoPtr)
    short theLAPType;
    Ptr protoPtr;
pascal short LAPCloseProtocol(theLAPType)
    short theLAPType;
pascal short LAPWrite(abRecord,async)
    ATLAPRecHandle abRecord;
    Boolean async;
pascal short LAPRead(abRecord,async)
    ATLAPRecHandle abRecord;
    Boolean async;
pascal short LAPRdCancel(abRecord)
    ATLAPRecHandle abRecord;

/* Datagram Delivery Protocol */

pascal short DDPOpenSocket(theSocket,sktListener)
    short *theSocket;
    Ptr sktListener;
pascal short DDPCloseSocket(theSocket)
    short theSocket;
pascal short DDPWrite(abRecord,doChecksum,async)
    ATDDPRecHandle abRecord;
    Boolean doChecksum,async;
pascal short DDPRead(abRecord,retCksumErrs,async)
    ATDDPRecHandle abRecord;
    Boolean retCksumErrs;
    Boolean async;
pascal short DDPRdCancel(abRecord)
    ATDDPRecHandle abRecord;

```

```

/* AppleTalk Transaction Protocol */

pascal short ATPLoad();
pascal short ATPUnload();
pascal short ATPOpenSocket(addrRcvd, atpSocket)
    AddrBlock addrRcvd;
    short *atpSocket;
pascal short ATPCloseSocket(atpSocket)
    short atpSocket;
pascal short ATPSndRequest(abRecord, async)
    ATATPRecHandle abRecord;
    Boolean async;
pascal short ATPRequest(abRecord, async)
    ATATPRecHandle abRecord;
    Boolean async;
pascal short ATPReqCancel(abRecord, async)
    ATATPRecHandle abRecord;
    Boolean async;
pascal short ATPGetRequest(abRecord, async)
    ATATPRecHandle abRecord;
    Boolean async;
pascal short ATPSndRsp(abRecord, async)
    ATATPRecHandle abRecord;
    Boolean async;
pascal short ATPAddRsp(abRecord)
    ATATPRecHandle abRecord;
pascal short ATPResponse(abRecord, async)
    ATATPRecHandle abRecord;
    Boolean async;
pascal short ATPRspCancel(abRecord, async)
    ATATPRecHandle abRecord;
    Boolean async;

/* Name-Binding Protocol */

pascal short NBPRegister(abRecord, async)
    ATNBPRECHandle abRecord;
    Boolean async;
pascal short NBPLookup(abRecord, async)
    ATNBPRECHandle abRecord;
    Boolean async;
pascal short NBPExtract(theBuffer, numInBuf, whichOne, abEntity, address)
    Ptr theBuffer;
    short numInBuf;
    short whichOne;
    EntityName *abEntity;
    AddrBlock *address;
pascal short NBPConfirm(abRecord, async)
    ATNBPRECHandle abRecord;
    Boolean async;
pascal short NBPRemove(abEntity)
    EntityPtr abEntity;
pascal short NBPLoad();
pascal short NBPUnload();

```



```
/* Miscellaneous Routines */
```

```
pascal void RemoveHdlBlocks();  
pascal short GetNodeAddress(myNode, myNet)  
    short *myNode,  
    short *myNet;  
pascal Boolean IsMPPOpen();  
pascal Boolean IsATPOpen();
```

Description The AppleTalk Manager provides an interface to the .MPP and .ATP AppleTalk device drivers in the 128K ROM.

For more detailed information, see the AppleTalk Manager chapter of *Inside Macintosh*.

Note Because C does not have variant records like Pascal, some Pascal records in *Inside Macintosh* are represented by more than one C typedef in this interface.

Controls—Control Manager

Synopsis

```
#include <Types.h>
#include <QuickDraw.h>
#include <Controls.h>

/* Control Definition Procedures IDs */

#define pushButProc      0
#define checkBoxProc    1
#define radioButProc    2
#define useWFont        8
#define scrollBarProc    16

/* FindControl Result Codes */

#define inButton         10
#define inCheckBox       11
#define inUpButton      20
#define inDownButton    21
#define inPageUp        22
#define inPageDown      23
#define inThumb         129

/* DragControl Axis Constraints */

#define noConstraint     0
#define hAxisOnly       1
#define vAxisOnly       2

/* Messages to Control Definition Function */

#define drawCntl        0
#define testCntl        1
#define calcCRgns      2
#define initCntl        3
#define dispCntl        4
#define posCntl         5
#define thumbCntl       6
#define dragCntl        7
#define autoTrack       8

typedef struct ControlRecord {
    struct ControlRecord **nextControl;
    struct GrafPort      *ctrlOwner;
    Rect                 ctrlRect;
    unsigned char         ctrlVis;
    unsigned char         ctrlHilite;
    short                 ctrlValue;
}
```

```

    short          contrlMin;
    short          contrlMax;
    Handle         contrlDefProc;
    Handle         contrlData;
    ProcPtr        contrlAction;
    long           contrlRfCon;
    Str255         contrlTitle;
} ControlRecord, *ControlPtr, **ControlHandle;

/* Initialization and Allocation */

ControlHandle NewControl(theWindow, boundsRect, title, visible, value,
    min, max, procID, refCon)
    struct GrafPort *theWindow;
    Rect             *boundsRect;
    char             *title;
    Boolean          visible;
    short            value;
    short            min;
    short            max;
    short            procID;
    long             refCon;
pascal ControlHandle GetNewControl(controlID, theWindow)
    short            controlID;
    struct GrafPort *theWindow;
pascal void DisposeControl(theControl)
    ControlHandle    theControl;
pascal void KillControls(theWindow)
    struct GrafPort *theWindow;

/* Control Display */

void SetCTitle(theControl, title)
    ControlHandle    theControl;
    char             *title;
void GetCTitle(theControl, title)
    ControlHandle    theControl;
    char             *title;
pascal void HideControl(theControl)
    ControlHandle    theControl;
pascal void ShowControl(theControl)
    ControlHandle    theControl;
pascal void DrawControls(theWindow)
    struct GrafPort *theWindow;
pascal void DrawControl(theControl)
    ControlHandle    theControl;
pascal void HiliteControl(theControl, hiliteState)
    ControlHandle    theControl;
    short            hiliteState;
pascal void UpdtControl(theWindow, updateRgn)
    struct GrafPort *theWindow;
    RgnHandle        updateRgn;

```

```

/* Mouse Location */

short FindControl(thePoint,theWindow,whichControl)
    Point          *thePoint;
    struct GrafPort *theWindow;
    ControlHandle   *whichControl;
short TrackControl(theControl,startPt,actionProc)
    ControlHandle   theControl;
    Point           *startPt;
    ProcPtr         actionProc;
short TestControl(theControl,thePoint)
    ControlHandle   theControl;
    Point           *thePoint;

/* Control Movement and Sizing */

pascal void MoveControl(theControl,h,v)
    ControlHandle   theControl;
    short           h;
    short           v;
void DragControl(theControl,startPt,limitRect,slopRect,axis)
    ControlHandle   theControl;
    Point           *startPt;
    Rect            *limitRect;
    Rect            *slopRect;
    short           axis;
pascal void SizeControl(theControl,w,h)
    ControlHandle   theControl;
    short           w;
    short           h;

/* Control Setting and Range */

pascal void SetCtlValue(theControl,theValue)
    ControlHandle theControl;
    short theValue;
pascal short GetCtlValue(theControl)
    ControlHandle theControl;
pascal void SetCtlMin(theControl,minValue)
    ControlHandle theControl;
    short minValue;
pascal short GetCtlMin(theControl)
    ControlHandle theControl;
pascal void SetCtlMax(theControl,maxValue)
    ControlHandle theControl;
    short maxValue;
pascal short GetCtlMax(theControl)
    ControlHandle theControl;

/* Miscellaneous Routines */

pascal void SetCRefCon(theControl,data)
    ControlHandle theControl;

```

```

    long data;
pascal long GetCRefCon(theControl)
    ControlHandle theControl;
pascal void SetCtlAction(theControl,actionProc)
    ControlHandle theControl;
    ProcPtr actionProc;
pascal ProcPtr GetCtlAction(theControl)
    ControlHandle theControl;

```

User routines

```

pascal void MyAction();
pascal void MyAction(theControl,partCode)
    ControlHandle theControl;
    short partCode;
pascal long MyControl(varCode,theControl,message,param)
    short varCode;
    ControlHandle theControl;
    short message;
    long param;

```

Description

The Control Manager provides routines for creating and manipulating controls (for example, buttons and scroll bars).

For more detailed information, see the Control Manager chapter of *Inside Macintosh*.

Desk—Desk Manager

Synopsis

```
#include <Types.h>
#include <Desk.h>

#define accEvent 64
#define accRun 65
#define accCursor 66
#define accMenu 67
#define accUndo 68
#define accCut 70
#define accCopy 71
#define accPaste 72
#define accClear 73

/* Opening and Closing Desk Accessories */

short OpenDeskAcc(theAcc)
    char *theAcc;
pascal void CloseDeskAcc(refNum)
    short refNum;

/* Handling Events in Desk Accessories */

pascal void SystemClick(theEvent,theWindow)
    struct EventRecord *theEvent;
    struct GrafPort *theWindow;
pascal Boolean SystemEdit(editCmd)
    short editCmd;

/* Performing Periodic Actions */

pascal void SystemTask();

/* Advanced Routines */

pascal Boolean SystemEvent(theEvent)
    struct EventRecord *theEvent;
pascal void SystemMenu(menuResult)
    long menuResult;
```

Description

The Desk Manager supports desk accessories.

For more detailed information, see the Desk Manager chapter of *Inside Macintosh*.

Note

Desk accessories do not have an A5 global area. Therefore all of the code for a desk accessory must reside in a single segment, no global variables may be declared, and no string constants may be used.

Warning

The names of desk accessories start with a null byte. The output parameter from `GetMenuItem` will return a string that begins with a null byte when a desk accessory is selected from the Apple menu. `OpenDeskAcc` skips over this null byte (if present) when interpreting its parameter.

Devices—device Manager

Synopsis

```
#include <Types.h>
#include <OSUtils.h>
#include <Windows.h>
#include <Files.h>
#include <Devices.h>
#define fsAtMark      0
#define fsCurPerm    0
#define fsRdPerm      1
#define fsWrPerm      2
#define fsRdWrPerm    3
#define fsRdWrShPerm  4

/* Chooser Message Values */

#define newSelMsg      12 /* new user selections have been made */
#define fillListMsg    13 /* fill the list with choices to be made */
#define getSelMsg      14 /* mark one or more choices as selected */
#define selectMsg      15 /* a choice has actually been made */
#define deselectMsg    16 /* a choice has been canceled */
#define terminateMsg   17 /* lets device package clean up */
#define buttonMsg      19 /* */

/* Caller Values */

#define chooserID      1 /* caller value for the Chooser */

typedef struct CntrlParam {
    struct QElem *qLink; /* next queue entry */
    short qType; /* queue type */
    short ioTrap; /* routine trap */
    Ptr ioCmdAddr; /* routine address */
    ProcPtr ioCompletion; /* completion routine */
    OSErr ioResult; /* result code */
    char *ioNamePtr; /* driver name */
    short ioVRefNum; /* volume reference or drive number */
    short ioCRefNum; /* driver reference number */
    short csCode; /* word for control status code */
    short csParam[11] /* operation-defined parameters */
} CntrlParam;

typedef struct DCtlEntry {
    Ptr dCtlDriver; /* ptr to ROM or handle to RAM driver */
    short dCtlFlags; /* flags */
    QHdr dCtlQHdr; /* driver's I/O queue */
    long dCtlPosition; /* byte pos used by read and write */
    Handle dCtlStorage; /* handle to RAM driver's storage */
    short dCtlRefNum; /* driver's reference number */
    long dCtlCurTicks; /* counter for timing system task calls */
}
```



```

    struct GrafPort *dCtlWindow; /* ptr to driver's window (if any) */
    short          dCtlDelay;    /* # of ticks between sysTask calls */
    short          dCtlEMask;    /* desk accessory event mask */
    short          dCtlMenu;     /* menu ID of menu associated w/driver */
} DCtlEntry, *DCtlPtr, **DCtlHandle;

/* High-Level Routines */

OSErr OpenDriver(name, refNum)
    char *name;
    short *refNum;
OSErr CloseDriver(refNum)
    short refNum;
OSErr FSRead(refnum, count, buffPtr)
    short refnum;
    long *count;
    Ptr buffPtr;
OSErr FSWrite(refnum, count, buffPtr)
    short refnum;
    long *count;
    Ptr buffPtr;
OSErr Control(refNum, csCode, csParamPtr)
    short refNum;
    short csCode;
    Ptr csParamPtr;
OSErr Status(refNum, csCode, csParamPtr)
    short refNum;
    short csCode;
    Ptr csParamPtr;
OSErr KillIO(refNum)
    short refNum;

/* Low-Level Routines */

OSErr PBOpen(paramBlock, async)
    struct ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBClose(paramBlock, async)
    struct ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBRead(paramBlock, async)
    struct ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBWrite(paramBlock, async)
    struct ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBControl(paramBlock, async)
    CntrlParam *paramBlock;
    Boolean async;
OSErr PBStatus(paramBlock, async)
    CntrlParam *paramBlock;
    Boolean async;
OSErr PBKillIO(paramBlock, async)

```

```

    CntrlParam *paramBlock;
    Boolean async;

    /* Accessing a Driver's I/O Queue */

    DCtlHandle GetDCtlEntry(refNum)
        short refNum;

```

User routines pascal OSErr Device(message, caller, objName, zoneName, p1, p2)

```

    short message;
    short caller;
    char *objName;
    char *zoneName;
    long p1, p2;

```

Description The Device Manager controls the exchange of information between applications and devices.

For more detailed information, see the Device Manager chapter of *Inside Macintosh*.

Warning The parameters objName and zoneName to the user routine Device must be Pascal strings.

Dialogs—Dialog Manager

Synopsis

```
#include <Types.h>
#include <QuickDraw.h>
#include <Windows.h>
#include <Dialogs.h>

/* Item Types */

#define ctrlItem      4 /* add to following four constants */
#define btnCtrl      0 /* standard button control */
#define chkCtrl      1 /* standard check box control */
#define radCtrl      2 /* standard "radio button" control */
#define resCtrl      3 /* control defined in control template */
#define statText     8 /* static text */
#define editText    16 /* editable text (dialog only) */
#define iconItem     32 /* icon */
#define picItem      64 /* Quickdraw picture */
#define userItem      0 /* application-defined item (dialog only) */
#define itemDisable 128 /* add to any of above to disable */

/* Item Numbers of OK and Cancel Buttons */

#define ok            1 /* OK button is first by convention */
#define cancel        2 /* Cancel button is second by convention */

/* Resource IDs of Alert Icons */

#define stopIcon      0
#define noteIcon      1
#define cautionIcon   2
typedef WindowPtr DialogPtr;

typedef struct DialogRecord {
    WindowRecord window;
    Handle items;
    struct TRec ** textH;
    short editField;
    short editOpen;
    short aDefItem;
} DialogRecord, *DialogPeek;

typedef struct DialogTemplate {
    Rect boundsRect;
    short procID;
    Boolean visible;
    Boolean filler1;
    Boolean goAwayFlag;
    Boolean filler2;
```

```

    long    refCon;
    short   itemsID;
    Str255  title;
} DialogTemplate, *DialogTPtr, **DialogTHndl;

typedef short StageList;

typedef struct AlertTemplate {
    Rect      boundsRect;
    short     itemsID;
    StageList stages;
} AlertTemplate, *AlertTPtr, **AlertTHndl;

/* Initialization */

pascal void InitDialogs(resumeProc)
    ProcPtr resumeProc;
pascal void ErrorSound(soundProc)
    ProcPtr soundProc;
void SetDAFont(fontNum)
    short fontNum;

/* Creating and Disposing of Dialogs */

DialogPtr NewDialog(dStorage, boundsRect, title, visible, procID, behind,
    goAwayFlag, refCon, items)
    Ptr dStorage;
    Rect *boundsRect;
    char *title;
    Boolean visible;
    short procID;
    WindowPtr behind;
    Boolean goAwayFlag;
    long refCon;
    Handle items;
pascal DialogPtr GetNewDialog(dialogID, dStorage, behind)
    short dialogID;
    Ptr dStorage;
    WindowPtr behind;
pascal void CloseDialog(theDialog)
    DialogPtr theDialog;
pascal void DisposDialog(theDialog)
    DialogPtr theDialog;
pascal void CouldDialog(dialogID)
    short dialogID;
pascal void FreeDialog(dialogID)
    short dialogID;

/* Handling Dialog Events */

pascal void ModalDialog(filterProc, itemHit)
    ProcPtr filterProc;
    short *itemHit;
pascal Boolean IsDialogEvent(theEvent)

```

```

    struct EventRecord *theEvent;
pascal Boolean DialogSelect(theEvent,theDialog,itemHit)
    struct EventRecord *theEvent;
    DialogPtr *theDialog;
    short *itemHit;
void DlgCut(theDialog)
    DialogPtr theDialog;
void DlgCopy(theDialog)
    DialogPtr theDialog;
void DlgPaste(theDialog)
    DialogPtr theDialog;
void DlgDelete(theDialog)
    DialogPtr theDialog;
pascal void DrawDialog(theDialog)
    DialogPtr theDialog;
pascal void UpdtDialog(theDialog,updateRgn)
    DialogPtr theDialog;
    RgnHandle updateRgn;

/* Invoking Alerts */

pascal short Alert(alertID,filterProc)
    short alertID;
    ProcPtr filterProc;
pascal short StopAlert(alertID,filterProc)
    short alertID;
    ProcPtr filterProc;
pascal short NoteAlert(alertID,filterProc)
    short alertID;
    ProcPtr filterProc;
pascal short CautionAlert(alertID,filterProc)
    short alertID;
    ProcPtr filterProc;
pascal void CouldAlert(alertID)
    short alertID;
pascal void FreeAlert(alertID)
    short alertID;

/* Manipulating Items in Dialogs and Alerts */

void ParamText(param0,param1,param2,param3)
    char *param0;
    char *param1;
    char *param2;
    char *param3;
pascal void GetDItem(theDialog,itemNo,itemType,item,box)
    DialogPtr theDialog;
    short itemNo;
    short *itemType;
    Handle *item;
    Rect *box;
pascal void SetDItem(theDialog,itemNo,type,item,box)
    DialogPtr theDialog;

```

```

    short itemNo;
    short type;
    Handle item;
    Rect *box;
pascal void HideDItem(theDialog, itemNo)
    DialogPtr theDialog;
    short itemNo;
pascal void ShowDItem(theDialog, itemNo)
    DialogPtr theDialog;
    short itemNo;
int FindDItem(theDialog, thePt)
    DialogPtr theDialog;
    Point *thePt;
void GetIText(item, text)
    Handle item;
    char *text;
void SetIText(item, text)
    Handle item;
    char *text;
pascal void SelIText(theDialog, itemNo, strtSel, endSel)
    DialogPtr theDialog;
    short itemNo;
    short strtSel;
    short endSel;
short GetAlrtStage();
void ResetAlrtStage();

```

User routines

```

pascal void MyItem(theWindow, itemNo)
    WindowPtr theWindow;
    short itemNo;
pascal void MySound(soundNo)
    short soundNo;
pascal Boolean MyFilter(theDialog, theEvent, itemHit)
    DialogPtr theDialog;
    struct EventRecord *theEvent;
    short *itemHit;

```

Description

The Dialog Manager supports dialog boxes and the alert mechanism. For more detailed information, see the Dialog Manager chapter of *Inside Macintosh*.

Note

StageList is defined as a short, rather than specifying the bits.

DiskInit--Disk Initialization Package

Synopsis

```
#include <Types.h>
#include <DiskInit.h>

/* Data Types */

typedef struct HFSDefaults {
    char    sigWord[2]; /* signature word */
    long    abSize;      /* allocation block size in bytes */
    long    clpSize;     /* clump size in bytes */
    long    nxFreeFN;    /* next free file number */
    long    btClpSize;   /* B*-Tree clump size in bytes */
    short   rsrv1;       /* reserved */
    short   rsrv2;       /* reserved */
    short   rsrv3;       /* reserved */
} HFSDefaults;

/* Routines */

void DILoad();
void DIUnload();
short DIBadMount (where, evtMessage)
    Point *where;
    long evtMessage;
OSErr DIFormat (drvNum)
    short drvNum;
OSErr DIVERify (drvNum)
    short drvNum;
OSErr DIZero (drvNum, volName)
    short drvNum;
    char *volName;
```

Description

The Disk Initialization Package found in the system-resource file initializes disks, formats the disk medium, and places appropriate file-directory structures on the disk. For more detailed information, see the Disk Initialization Package chapter of *Inside Macintosh*.

Disks—Disk Driver

Synopsis

```
#include <Types.h>
#include <Disks.h>

/* Positioning Modes */

#define fsAtMark      0 /* at current sector */
#define fsFromStart  1 /* offset relative to beginning of file */
#define fsFromMark   3 /* offset relative to current mark */
#define rdVerify     64 /* added to above for read-verify */

/* Data Types */

typedef enum {sony, hard20} DriveKind;
typedef struct DrvStsSony {
    short    track;          /* current track */
    char     writeProt;      /* bit 7 = 1 if volume is locked */
    char     diskInPlace;    /* disk in place */
    char     installed;      /* drive installed */
    char     sides;          /* bit 7 = 0 if 1-sided drive */
    struct QElem *qLink;     /* next queue entry */
    short    qType;          /* not used */
    short    dQDrive;        /* drive number */
    short    dQRefNum;       /* driver reference number */
    short    dQFSID;         /* file-system identifier */
    char     twoSideFmt;     /* -1 if 2-sided disk */
    char     needsFlush;     /* reserved */
    short    diskErrs;       /* error count */
} DrvStsSony;

typedef struct DrvStsHard20 {
    short    track;          /* current track */
    char     writeProt;      /* bit 7 = 1 if volume is locked */
    char     diskInPlace;    /* disk in place */
    char     installed;      /* drive installed */
    char     sides;          /* bit 7 = 0 if 1-sided drive */
    struct QElem *qLink;     /* next queue entry */
    short    qType;          /* not used */
    short    dQDrive;        /* drive number */
    short    dQRefNum;       /* driver reference number */
    short    dQFSID;         /* file-system identifier */
    short    DrvSize;        /* drive block size low word */
    short    DrvS1;          /* drive block size high word */
    short    DrvType;        /* 1 for Hard Disk 20 */
    short    DrvManf;        /* 1 for Apple Computer, Inc. */
    short    DrvChar;        /* 230 (0xe6) for Hard Disk 20 */
    char     DrvMisc;        /* 0--reserved */
} DrvStsHard20;
```



```

/* Routines */

OSErr DiskEject(drvNum)
    short drvNum;
OSErr SetTagBuffer(buffPtr)
    Ptr buffPtr;
OSErr DriveStatus(drvNum, status)
    short drvNum;
    DrvSts *status;

```

Description

The Disk Driver is a Macintosh device driver used for storing and retrieving information on Macintosh 3.5-inch disk drives.

For more detailed information, see the Disk Driver chapter of *Inside Macintosh*.

Errors—System Error Handler

Synopsis

```
#include <Errors.h>

/* AppleTalk Manager Errors */

#define sktClosedErr (-3109)
#define atPlenErr (-3106)
#define readQErr (-3105)
#define extractErr (-3104)
#define ckSumErr (-3103)
#define noMPPErr (-3102)
#define buf2SmallErr (-3101)
#define noRelErr (-1101)
#define nbpNISErr (-1029)

/* Resource Manager Errors */

#define mapReadErr (-199)
#define resAttrErr (-198)
#define rmvRefFailed (-197)
#define rmvResFailed (-196)
#define addRefFailed (-195)
#define addResFailed (-194)
#define resFNotFound (-193)
#define resNotFound (-192)

/* File Manager Errors */

#define fsDSIntErr (-127)
#define wrgVolTypErr (-123)
#define badMovErr (-122)
#define tmwdoErr (-121)
#define dirNFErr (-120)

/* Memory Manager Errors */

#define memLockedErr (-117) /* block is locked */
#define memSCErr (-116)
#define memBCErr (-115)
#define memPCErr (-114)
#define memAZErr (-113)
#define memPurErr (-112) /* attempt to purge a locked block */
#define memWZErr (-111) /* attempt to operate on a free block */
#define memAdrErr (-110)
#define nilHandleErr (-109) /* nil master pointer */
#define memFullErr (-108) /* not enough room in zone */
#define noTypeErr (-102) /* no data of the requested type */
#define noScrapErr (-100) /* desk scrap isn't initialized */
```

```
/* More AppleTalk Manager Errors */
```

```
#define memROZErr      (-99)
#define portInUse      (-97)
#define portNotCf      (-98)
#define excessCollsns  (-95)
#define LAPPProtErr    (-94)
#define noBridgeErr    (-93)
#define ddpLenErr      (-92)
#define ddpSktErr      (-91)
#define breakRecd      (-90)
```

```
/* Miscellaneous Errors */
```

```
#define rcvrErr        (-89)
#define prInitErr      (-88) /* validity status is not 0xa8 */
#define prWrErr        (-87) /* parameter RAM written did not verify */
#define clkWrErr       (-86) /* time written did not verify */
#define clkRdErr       (-85) /* unable to read clock */
#define firstDskErr    (-84) /* first of the range of disk errors */
#define sectNFErr      (-81) /* can't find sector */
#define seekErr        (-80) /* hardware error */
#define spdAdjErr      (-79) /* hardware error */
#define twoSideErr     (-78) /* tried to read side 2 on 1-sided drive */
#define initIWMErr     (-77) /* hardware error */
#define tk0BadErr      (-76) /* hardware error */
#define cantStepErr    (-75) /* hardware error */
#define cantStepErr    (-75)
#define wrUnderrun     (-74)
#define badDBtSlp      (-73)
#define badDCksum      (-72)
#define noDtAmkErr     (-71) /* can't find data mark */
#define badBtSlpErr    (-70) /* bad address mark */
#define badCksmErr     (-69) /* bad address mark */
#define dataVerErr     (-68) /* read-verify failed */
#define noAdrMkErr     (-67) /* can't find an address mark */
#define fontSubErr     (-66)
#define noNybErr       (-66) /* disk is probably blank */
#define offLinErr      (-65) /* no disk in drive */
#define fontNotDeclared (-65)
#define fontDecError   (-64)
#define lastDskErr     (-64) /* last of the range of disk errors */
#define noDriveErr     (-64) /* drive isn't connected */
#define wrPermErr      (-61) /* permission does not allow writing */
#define badMDBErr      (-60) /* master directory block bad, reinit */
#define fsRnErr        (-59) /* problem during rename */
#define extFSErr       (-58) /* external file system */
                          /* file-system identifier < 0 or */
                          /* path ref-num > 1024 */
#define noMacDskErr    (-57) /* volume lacks Mac-format directory */
#define nsDrvErr       (-56) /* no such drive in the drive queue */
#define volOnLinErr    (-55) /* volume already mounted and on-line */
#define permErr        (-54) /* permission doesn't allow writing */
#define volOffLinErr   (-53) /* volume not on-line */
```

```

#define gfpErr          (-52)
#define rfNumErr        (-51) /* bad reference number */
#define paramErr        (-50)

/* File Manager Errors */

#define opWrErr          (-49) /* only one writer allowed */
#define dupFNErr         (-48) /* file by that name already exists */
#define fBsyErr          (-47) /* one or more files are open */
#define vLckdErr         (-46) /* volume locked by software flag */
#define fLckdErr         (-45) /* file locked */
#define wPrErr           (-44) /* volume locked by hardware setting */
#define fnfErr           (-43) /* file not found */
#define tmfoErr          (-42) /* only 12 files can be open at once */
#define posErr           (-40) /* attempted to position before start */
#define eofErr           (-39) /* logical EOF reached during read */
#define fnOpnErr         (-38) /* file not open */
#define bdNamErr         (-37) /* bad filename or volume name--zero*/
                          /* length? */
#define ioErr            (-36) /* disk I/O error */
#define nsvErr           (-35) /* specified volume doesn't exist */
#define dskFulErr        (-34) /* all allocation blocks are full */
#define dirFulErr        (-33) /* file directory full */
#define notOpenErr       (-28) /* driver isn't open */
#define abortErr         (-27) /* I/O request aborted by KillIO */
#define dInstErr         (-26) /* couldn't find driver in resource file */
#define dRemoveErr       (-25) /* tried to remove an open driver */
#define closeErr         (-24)
#define openErr          (-23) /* requested r/w permission refused */
#define unitEmptyErr     (-22) /* reference number specifies nil handle */
#define badUnitErr       (-21) /* refNum doesn't match unit table */
#define writErr          (-20) /* driver can't respond to Write calls */
#define readErr          (-19) /* driver can't respond to Read calls */
#define statusErr        (-18) /* driver can't respond to Status call */
#define controlErr       (-17) /* driver can't respond to this call */

/* More Miscellaneous Errors */

#define SENoDB           (-8)
#define unimpErr         (-4)
#define corErr           (-3)
#define vTypErr          (-2) /* qType field isn't vType */
#define qErr             (-1) /* element not in specified queue */
#define noErr            0 /* no error */
#define dsBusErr         1 /* bus Error */
#define evtNotEnb        1
#define swOverrunErr     1 /* set if software overrun error */
#define scCommErr        2 /* breakdown in SCSI protocols: */
#define dsAddressErr     2 /* address error */
#define dsIllInstErr     3 /* illegal instruction */
#define dsZeroDivErr     4 /* zero divide */
#define scBadParmsErr    4 /* unrecognized instruction in TIB */
#define scPhaseErr       5

```

```

#define dsChkErr      5 /* check exception */
#define dsOvflowErr   6 /* TrapV exception */
#define scCompareErr  6 /* data comparison during read */
#define dsPrivErr     7 /* privilege violation */
#define dsTraceErr    8 /* trace exception */
#define dsLineAErr    9 /* line 1010 exception */
#define dsLineFErr   10 /* line 1111 exception */
#define dsMiscErr     11 /* miscellaneous exception */
#define dsCoreErr     12 /* unimplemented core routine */
#define dsIrqErr      13 /* spurious interrupt */
#define dsIOCoreErr   14 /* I/O system error */
#define dsLoadErr     15 /* segment loader error */
#define dsFPERR       16 /* floating-point error */
#define parityErr     16 /* set if parity error */
#define dsNoPackErr   17 /* can't load package 0 */
#define dsMemFullErr  25 /* out of memory */
#define dsFSERR       27 /* file map trashed */
#define dsStknHeap    28
#define dsReinsert    30
#define dsNotThe1     31

/* More Miscellaneous Errors */

#define menuPrgErr     84
#define hwOverrunErr   32 /* set if hardware overrun error */
#define framingErr     64 /* set if framing error */
#define dsSysErr       32767 /* system error */
void SysError(errorCode)
    short errorCode;

```

Description The System Error Handler is the part of the Macintosh Operating System that assumes control when a fatal error occurs.

For more detailed information, see the System Error Handler chapter of *Inside Macintosh*.

Events—Toolbox Event Manager

Synopsis

```
#include <Types.h>
#include <Events.h>

/* Event Codes */

#define nullEvent      0
#define mouseDown     1
#define mouseUp       2
#define keyDown       3
#define keyUp         4
#define autoKey       5
#define updateEvt     6
#define diskEvt       7
#define activateEvt   8
#define networkEvt    10
#define driverEvt     11
#define app1Evt       12
#define app2Evt       13
#define app3Evt       14
#define app4Evt       15

/* Masks for Keyboard Event Message */

#define charCodeMask   0x000000FF
#define keyCodeMask    0x0000FF00

/* Masks for Forming Event Mask */

#define mDownMask      2
#define mUpMask        4
#define keyDownMask    8
#define keyUpMask     16
#define autoKeyMask    32
#define updateMask     64
#define diskMask       128
#define activMask      256
#define networkMask    1024
#define driverMask     2048
#define app1Mask       4096
#define app2Mask       8192
#define app3Mask       16384
#define app4Mask       (-32768)
#define everyEvent     (-1)

/* Modifier Flags in Event Record */

#define activeFlag     1
```

```

#define btnState      128
#define cmdKey        256
#define shiftKey      512
#define alphaLock     1024
#define optionKey     2048

/* Data Types */

typedef struct EventRecord {
    short      what;
    long       message;
    long       when;
    Point      where;
    short      modifiers;
} EventRecord;

typedef long KeyMap[4];

/* Accessing Events */

pascal Boolean GetNextEvent(eventMask,theEvent)
    short eventMask;
    EventRecord *theEvent;
pascal Boolean EventAvail(eventMask,theEvent)
    short eventMask;
    EventRecord *theEvent;

/* Reading the Mouse */

pascal void GetMouse(mouseLoc)
    Point *mouseLoc;
pascal Boolean Button();
pascal Boolean StillDown();
pascal Boolean WaitMouseUp();

/* Reading the Keyboard and Keypad */

pascal void GetKeys(theKeys)
    KeyMap theKeys;

/* Miscellaneous Routines */

pascal long TickCount();
long GetDblTime();
long GetCaretTime();

```

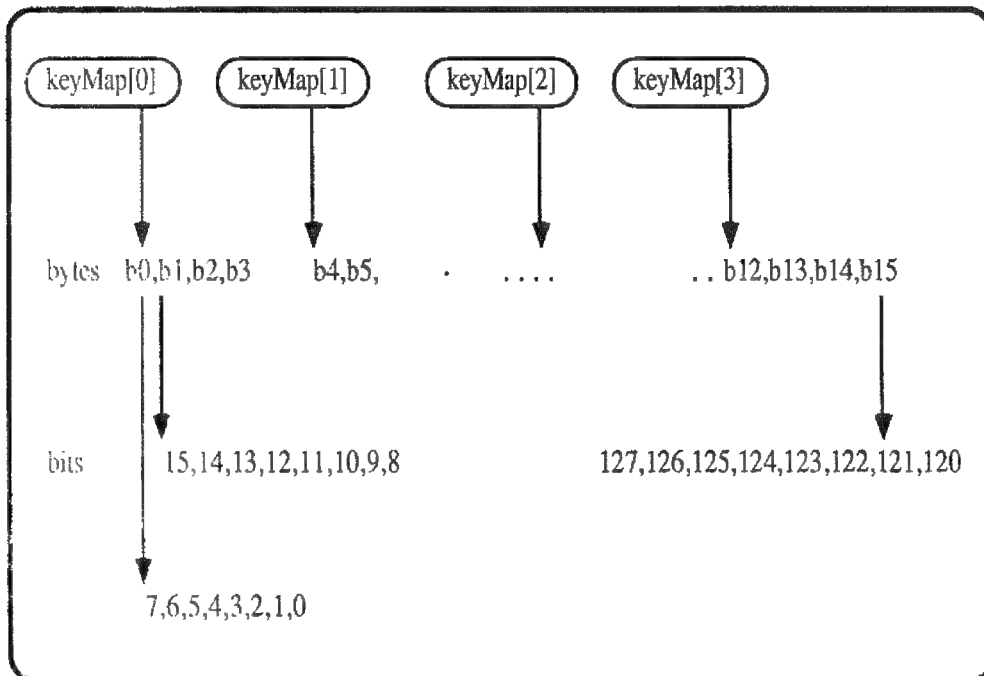
Description

The Toolbox Event Manager provides access to the Macintosh keyboard, keypad, and mouse. The keyboard bit map returned by the function `GetKeys()` is organized as shown in Figure 4-1. The bits aren't numbered as you might expect. Here is the actual numbering scheme, corresponding to the key code numbers given in the figure "Key Codes" in the Toolbox Event Manager chapter of *Inside Macintosh*.

Figure 4-1

The keyboard bit map

Figure 4-1. The Keyboard Bit Map



For more detailed information, see the Toolbox Event Manager chapter of *Inside Macintosh*.

Files—File Manager

Synopsis

```
#include <Types.h>
#include <OSUtils.h>
#include <Files.h>

/* Flags in File Information Used by the Finder */

#define fOnDesk      1 /* set if file is on desktop (HFS only) */
#define fHasBundle  8192 /* set if file has a bundle. */
#define fInvisible  16384 /* set if file's icon is invisible */
#define fTrash      (-3) /* file is in trash window */
#define fDesktop    (-2) /* file is on desktop */
#define fDisk        0 /* file is in disk window */

/* Values for Requesting Read/Write Permission */

#define fsCurPerm    0 /* whatever is currently allowed */
#define fsRdPerm     1 /* request to read only */
#define fsWrPerm     2 /* request to write only */
#define fsRdWrPerm   3 /* request to read and write */
#define fsRdWrShPerm 4 /* request shared read and write */

/* Positioning Modes */

#define fsAtMark      0 /* at current position of mark */
                        /* (posOff or ioPosOffset ignored) */
#define fsFromStart   1 /* offset relative to beginning of file */
#define fsFromLEOF    2 /* offset relative to logical EOF */
#define fsFromMark    3 /* offset relative to current mark */
#define rdVerify      64 /* added to above for read-verify */
#define ioDirFlg      4
#define ioDirMask     (1 << 4)
#define fsRtParID     1
#define fsRtDirID     2

/* Data Types */

typedef struct FInfo {
    OSType  fdType; /* the type of the file */
    OSType  fdCreator; /* file's creator */
    short   fdFlags; /* flags: hasBundle, invisible, etc. */
    Point   fdLocation; /* file's location in window */
    short   fdFlDr; /* folder containing file */
} FInfo;

typedef struct FXInfo {
    short   ffIconID; /* Icon ID */
    short   ffUnused[4]; /* reserved */
}
```

```

    short          ffComment;      /* comment ID */
    long           ffPutAway;      /* home directory ID */
} FXInfo ;

typedef struct DInfo {
    Rect           frRect;         /* folder's rectangle */
    unsigned short frFlags;        /* flags */
    Point          frLocation;     /* folder's location */
    short          frView;         /* folder's view */
} DInfo;

typedef struct DXInfo {
    Point          frScroll;       /* scroll position */
    long           frOpenChain;    /* directory ID chain of open folders */
    short          frUnused;       /* reserved */
    short          frComment;      /* comment ID */
    long           frPutAway;      /* directory ID */
} DXInfo;

typedef struct IOParam {
    struct QElem *qLink;          /* next queue entry */
    short         qType;          /* queue type */
    short         ioTrap;         /* routine trap */
    Ptr           ioCmdAddr;      /* routine address */
    ProcPtr       ioCompletion;   /* completion routine */
    OSERR         ioResult;       /* result code */
    StringPtr     ioNamePtr;      /* pathname */
    short         ioVRefNum;      /* volume refnum or drive number */
    short         ioRefNum;       /* path reference number */
    char          ioVersNum;      /* version number */
    char          ioPermssn;      /* read/write permission */
    Ptr           ioMisc;         /* miscellaneous (use for PBSetFVers) */
    Ptr           ioBuffer;       /* data buffer */
    long          ioReqCount;      /* requested number of bytes */
    long          ioActCount;      /* actual byte count completed */
    short         ioPosMode;      /* newline char and type of
                                   /* positioning */
    long          ioPosOffset;    /* size of positioning offset */
} IOParam;

typedef struct FileParam {
    struct QElem *qLink;          /* next queue entry */
    short         qType;          /* queue type */
    short         ioTrap;         /* routine trap */
    Ptr           ioCmdAddr;      /* routine address */
    ProcPtr       ioCompletion;   /* completion routine */
    OSERR         ioResult;       /* result code */
    StringPtr     ioNamePtr;      /* pathname */
    short         ioVRefNum;      /* volume refnum or drive number */
    short         ioFRefNum;      /* path reference number */
    char          ioFVersNum;     /* version number */
    char          filler1;        /* not used */
    short         ioFDirIndex;    /* sequence number of file */

```

```

    unsigned char    ioFlAttrib;    /* file attributes */
    unsigned char    ioFlVersNum;   /* version number */
    FInfo            ioFlFndrInfo;  /* information used by the Finder */
    unsigned long    ioFlNum;       /* file number */
    unsigned short   ioFlStBlk;     /* first block of data fork */
    long             ioFlLgLen;     /* logical EOF of data fork */
    long             ioFlPyLen;     /* physical EOF of data fork */
    unsigned short   ioFlRStBlk;    /* first block of resource fork */
    long             ioFlRLgLen;    /* logical EOF of resource fork */
    long             ioFlRPyLen;    /* physical EOF of resource fork */
    unsigned long    ioFlCrDat;     /* date and time of creation */
    unsigned long    ioFlMdDat;     /* date and time of last modification */
} FileParam;

```

```

typedef struct VolumeParam {
    struct QElem *qLink;           /* next queue entry */
    short          qType;          /* queue type */
    short          ioTrap;         /* routine trap */
    Ptr            ioCmdAddr;      /* routine address */
    ProcPtr        ioCompletion;   /* completion routine */
    OSErr          ioResult;       /* result code */
    StringPtr      ioNamePtr;      /* pathname */
    short          ioVRefNum;      /* volume refnum or drive number */
    long           filler2;        /* not used */
    short          ioVolIndex;     /* volume index */
    unsigned long  ioVCrDate;      /* date and time of initialization */
    unsigned long  ioVLSBkUp;      /* date and time of last volume backup */
    unsigned short ioVAtrb;        /* bit 15=1 if volume locked */
    unsigned short ioVNmFls;      /* number of files in file directory */
    unsigned short ioVDirSt;      /* first block of file directory */
    short          ioVBln;        /* number of blocks in file directory */
    unsigned short ioVNmAlBlks;    /* number of alloc blocks on volume */
    long           ioVALBlkSiz;    /* number of bytes per alloc block */
    long           ioVClpSiz;      /* number of bytes to allocate */
    unsigned short ioAlBlSt;      /* first block in volume block map */
    unsigned long  ioVNxtFNum;     /* next free file number */
    unsigned short ioVFrBlk;      /* number of free allocation blocks */
} VolumeParam;

```

```

typedef struct HIOParam {
    struct QElem *qLink;           /* next queue entry */
    short          qType;          /* queue type */
    short          ioTrap;         /* routine trap */
    Ptr            ioCmdAddr;      /* routine address */
    ProcPtr        ioCompletion;   /* completion routine */
    OSErr          ioResult;       /* result code */
    StringPtr      ioNamePtr;      /* pathname */
    short          ioVRefNum;      /* volume refnum or drive number */
    short          ioRefNum;       /* path reference number */
    char           ioVersNum;      /* version number */
    char           ioPermssn;      /* read/write permission */
    Ptr            ioMisc;         /* miscellaneous (use high byte for */
                                /* PBSetFVers) */
    Ptr            ioBuffer;       /* data buffer */
} HIOParam;

```

```

    long          ioReqCount;    /* requested number of bytes */
    long          ioActCount;    /* actual byte count completed */
    short         ioPosMode;     /* newline char and type of positioning */
    long          ioPosOffset;   /* size of positioning offset */
    short         filler1;       /* empty field */
} HIOParam;

typedef union ParamBlockRec {
    struct IOParam ioParam;
    struct FileParam fileParam;
    struct VolumeParam volumeParam;
} ParamBlockRec, *ParmBlkPtr;

typedef struct HFileParam {
    struct QElem *qLink;        /* next queue entry */
    short         qType;        /* queue type */
    short         ioTrap;       /* routine trap */
    Ptr           ioCmdAddr;     /* routine address */
    ProcPtr       ioCompletion;  /* completion routine */
    OSErr         ioResult;     /* result code */
    StringPtr     ioNamePtr;     /* pathname */
    short         ioVRefNum;     /* volume refnum or drive number */
    short         ioFRefNum;     /* path reference number */
    char          ioFVersNum;    /* version number */
    char          filler1;       /* not used */
    short         ioFDirIndex;   /* sequence number of file */
    char          ioFlAttrib;    /* file attributes */
    char          ioFlVersNum;   /* version number */
    FInfo         ioFlFndrInfo;  /* information used by the Finder */
    long          ioDirID;       /* directory ID */
    unsigned short ioFlStBlk;     /* first block of data fork */
    long          ioFlLgLen;      /* logical EOF of data fork */
    long          ioFlPyLen;      /* physical EOF of data fork */
    unsigned short ioFlRStBlk;    /* first block of resource fork */
    long          ioFlRLgLen;     /* logical EOF of resource fork */
    long          ioFlRPyLen;     /* physical EOF of resource fork */
    unsigned long  ioFlCrDat;     /* date and time of creation */
    unsigned long  ioFlMdDat;     /* date and time of last modification */
} HFileParam;

typedef struct HVolumeParam {
    struct QElem *qLink;        /* next queue entry */
    short         qType;        /* queue type */
    short         ioTrap;       /* routine trap */
    Ptr           ioCmdAddr;     /* routine address */
    ProcPtr       ioCompletion;  /* completion routine */
    OSErr         ioResult;     /* result code */
    StringPtr     ioNamePtr;     /* pathname */
    short         ioVRefNum;     /* volume refnum or drive number */
    long          filler2;       /* not used */
    short         ioVolIndex;    /* volume index */
    unsigned long  ioVCrDate;    /* date and time of initialization */
    unsigned long  ioVLsMod;     /* date and time of last modification */
    unsigned short ioVAtrb;      /* volume attributes */
}

```

```

    unsigned short    ioVNmFls;        /* number of files in file directory */
    short             ioVBitMap;        /* first block of volume bitmap */
    short             ioAllocPtr;       /* used internally */
    unsigned short    ioVNmAlBlks;     /* number of alloc blocks on volume */
    long              ioVA1BlkSiz;      /* number of bytes per alloc block */
    long              ioVC1pSiz;        /* number of bytes to Allocate */
    short             ioAlBlSt;         /* first block in volume block map */
    long              ioVNxtCNID;        /* next unused node ID */
    unsigned short    ioVFrBlk;        /* number of free allocation blocks */
    unsigned short    ioVSigWord;       /* volume signature */
    short             ioVDrvInfo;        /* drive number */
    short             ioVRefNum;        /* driver reference number */
    short             ioVFSID;          /* file-system identifier */
    unsigned long      ioVBkUp;         /* date and time of last backup */
    unsigned short    ioVSeqNum;        /* used internally */
    long              ioVWrCnt;         /* volume write count */
    long              ioVFilCnt;        /* number of files on volume */
    long              ioVDirCnt;        /* number of directories on volume */
    long              ioVFndrInfo[8];   /* info used by the Finder */
} HVolumeParam;

typedef union HParamBlockRec {
    struct HIOParam ioParam;
    struct HFileParam fileParam;
    struct HVolumeParam volumeParam;
} HParamBlockRec, *HParamBlkPtr;

typedef enum {hfileInfo,dirInfo} CInfoType;

typedef struct HFileInfo {
    struct QElem *qLink;                /* next queue entry */
    short          qType;                /* queue type */
    short          ioTrap;               /* routine trap */
    Ptr            ioCmdAddr;            /* routine address */
    ProcPtr        ioCompletion;         /* completion routine */
    OSErr          ioResult;             /* result code */
    StringPtr      ioNamePtr;            /* pathname */
    short          ioVRefNum;            /* volume refnum or drive number */
    short          ioFRefNum;            /* path reference number */
    char           ioFVersNum;           /* version number */
    char           filler1;              /* not used */
    short          ioFDirIndex;          /* sequence number of file */
    char           ioFlAttrib;           /* file attributes */
    char           filler2;              /* not used */
    FInfo          ioFlFndrInfo;         /* information used by the Finder */
    long           ioDirID;              /* file number */
    unsigned short ioFlStBlk;            /* first block of data fork */
    long           ioFlLgLen;            /* logical EOF of data fork */
    long           ioFlPyLen;            /* physical EOF of data fork */
    unsigned short ioFlRStBlk;           /* first block of resource fork */
    long           ioFlRLgLen;           /* logical EOF of resource fork */
    long           ioFlRPyLen;           /* physical EOF of resource fork */
    unsigned long  ioFlCrDat;            /* date and time of creation */
    unsigned long  ioFlMdDat;            /* date and time of last modification */

```

```

        unsigned long    ioFlBkDat;        /* date and time of last backup */
        FXInfo           ioFlXFndrInfo;    /* additional info for Finder */
        long             ioFlParID;        /* file's parent directory ID */
        long             ioFlClpSiz;       /* file's clump size */
    } HFileInfo;

typedef struct DirInfo {
    struct QElem *qLink;                /* next queue entry */
    short          qType;                /* queue type */
    short          ioTrap;               /* routine trap */
    Ptr            ioCmdAddr;            /* routine address */
    ProcPtr        ioCompletion;         /* completion routine */
    OSErr          ioResult;             /* result code */
    StringPtr      ioNamePtr;            /* pathname */
    short          ioVRefNum;            /* volume refnum or drive number */
    short          ioFRefNum;            /* path reference number */
    char           ioFVersNum;           /* version number */
    char           filler1;              /* not used */
    short          ioFDirIndex;          /* sequence number of file */
    char           ioFlAttrib;           /* file attributes */
    char           filler2;              /* not used */
    DInfo          ioDrUsrWds;           /* information used by the Finder */
    long           ioDrDirID;            /* directory ID */
    unsigned short ioDrNmFls;            /* number of files in directory */
    short          filler3[9];           /* not used */
    unsigned long  ioDrCrDat;            /* date and time of creation */
    unsigned long  ioDrMdDat;            /* date and time of last modification */
    unsigned long  ioDrBkDat;            /* date and time of last backup */
    DXInfo         ioDrFndrInfo;         /* additional info for Finder */
    long           ioDrParID;            /* file's parent directory ID */
} DirInfo;

typedef union CInfoPBRec {
    HFileInfo hfileInfo;
    DirInfo dirInfo;
} CInfoPBRec;

typedef struct CMovePBRec {
    struct QElem *qLink;                /* next queue entry */
    short          qType;                /* queue type */
    short          ioTrap;               /* routine trap */
    Ptr            ioCmdAddr;            /* routine address */
    ProcPtr        ioCompletion;         /* completion routine */
    OSErr          ioResult;             /* result code */
    StringPtr      ioNamePtr;            /* pathname */
    short          ioVRefNum;            /* volume refnum or drive number */
    long           filler1;              /* not used */
    StringPtr      ioNewName;            /* name of new directory */
    long           filler2;              /* not used */
    long           ioNewDirID;           /* directory ID of new directory */
    long           filler3[2];           /* not used */
    long           ioDirID;              /* directory ID of current directory */
} CMovePBRec, *CMovePBPtr;

```

```

typedef enum {hfileInfo, dirInfo} CInfoType;

typedef struct WDPBRec {
    struct QElem *qLink;          /* next queue entry */
    short          qType;         /* queue type */
    short          ioTrap;        /* routine trap */
    Ptr            ioCmdAddr;     /* routine address */
    ProcPtr        ioCompletion;  /* completion routine */
    OSErr          ioResult;      /* result code */
    StringPtr      ioNamePtr;     /* pathname */
    short          ioVRefNum;     /* volume refnum or drive number */
    short          filler1;       /* not used */
    short          ioWDIndex;     /* working directory index */
    long           ioWDPID;       /* working directory user identifier */
    short          ioWDVRefNum;   /* working directory's volume ref number */
    short          filler2[7];
    long           ioWDDirID;     /* working directory's directory ID */
} WDPBRec, *WDPBPtr;

typedef struct FCBPBBRec {
    struct QElem *qLink;          /* next queue entry */
    short          qType;         /* queue type */
    short          ioTrap;        /* routine trap */
    Ptr            ioCmdAddr;     /* routine address */
    ProcPtr        ioCompletion;  /* completion routine */
    OSErr          ioResult;      /* result code */
    StringPtr      ioNamePtr;     /* pathname */
    short          ioVRefNum;     /* volume refnum or drive number */
    short          ioRefNum;      /* path reference number */
    short          filler;
    short          filler1;       /* FCB index */
    long           ioFCBF1Nm;     /* file number */
    short          ioFCBFlags;
    unsigned short ioFCBStBlk;    /* first allocation block of file */
    long           ioFCBEOF;      /* logical EOF */
    long           ioFCBPLen;     /* physical EOF */
    long           ioFCBCrPs;     /* mark */
    short          ioFCBVRefNum;  /* volume reference number */
    long           ioFCBClpSiz;   /* file's clump size */
    long           ioFCBParID;    /* parent directory ID */
} FCBPBBRec, *FCBPBBPtr;

typedef struct VCB {
    struct QElem *qLink;          /* next queue entry */
    short          qType;         /* not used */
    short          vcbFlags;      /* bit 15=1 if dirty */
    unsigned short vcbSigWord;    /* always 0xd2d7 */
    unsigned long  vcbCrDate;     /* date volume was initialized */
    unsigned long  vcbLsMod;      /* date of last backup */
    short          vcbAtrb;       /* volume attributes */
    unsigned short vcbNmFls;      /* number of files in directory */
    short          vcbVBMSt;      /* directory's first block */
    short          vcbAllocPtr;   /* length of file directory */
    unsigned short vcbNmAlBks;    /* number of allocation blocks */
}

```



```

long          vcbAlBlkSiz;    /* size of allocation blocks */
long          vcbClpSiz;     /* number of bytes to Allocate */
short        vcbAlBlst;     /* first block in block map */
long          vcbNxtCNID;    /* next unused file number */
unsigned short vcbFreeBks;   /* number of unused blocks */
String(27)    vcbVN;        /* volume name */
short        vcbDrvNum;     /* drive number */
short        vcbDRefNum;    /* driver reference number */
short        vcbFSID;       /* file-system identifier */
short        vcbVRefNum;    /* volume reference number */
Ptr          vcbMAdr;       /* location of block map */
Ptr          vcbBufAdr;     /* location of volume buffer */
short        vcbMLen;       /* number of bytes in block map */
short        vcbDirIndex;   /* used internally */
short        vcbDirBlk;     /* used internally */
unsigned long vcbVolBkUp;    /* date and time of last backup */
unsigned short vcbVSeqNum;   /* used internally */
long         vcbWrCnt;       /* volume write count */
long         vcbXTClpSiz;    /* clump size of extents tree file */
long         vcbCTClpSiz;    /* clump size of catalog tree file */
unsigned short vcbNmRtDirs;  /* number of directories in root */
long         vcbFilCnt;     /* number of files on volume */
long         vcbDirCnt;     /* number of directories on volume */
long         vcbFndrInfo[8]; /* used by Finder */
unsigned short vcbVCSiz;    /* used internally */
unsigned short vcbVBMCSiz;  /* used internally */
unsigned short vcbCtlCSiz;  /* used internally */
unsigned short vcbXTAlBlks;  /* size in blocks of extents tree file */
unsigned short vcbCTAlBlks;  /* size in blocks of catalog tree file */
short         vcbXTRef;     /* path reference number for extents */
/* tree file */
short         vcbCTRef;     /* path ref number of catalog tree file */
Ptr          vcbCtlBuf;     /* pointer to extents and catalog caches */
long         vcbDirIDM;     /* directory last searched */
short        vcbOffsM;      /* offspring index at last search */
} VCB;

typedef struct DrvQEl {
    struct QElem *qLink;    /* next queue entry */
    short        qType;     /* queue type */
    short        dQDrive;   /* drive number */
    short        dQRefNum;  /* drive reference number */
    short        dQFSID;    /* file system identifier */
    unsigned short dQDrvSz;  /* number of logical blocks */
    unsigned short dQDrvSz2;
} DrvQEl, *DrvQElPtr;

/* --- High-Level Routines ----- */

/* Accessing Volumes */

OSErr GetVInfo(drvNum, volName, vRefNum, freeBytes)
    short drvNum;

```

```

    char *volName;
    short *vRefNum;
    long *freeBytes;
OSErr GetVRefNum(pathRefNum, vRefNum)
    short pathRefNum;
    short *vRefNum;
OSErr GetVol(volName, vRefNum)
    char *volName;
    short *vRefNum;
OSErr SetVol(volName, vRefNum)
    char *volName;
    short vRefNum;
OSErr FlushVol(volName, vRefNum)
    char *volName;
    short vRefNum;
OSErr UnmountVol(volName, vRefNum)
    char *volName;
    short vRefNum;
OSErr Eject(volName, vRefNum)
    char *volName;
    short vRefNum;

/* Changing File Contents */

OSErr FSOpen(fileName, vRefNum, refNum)
    char *fileName;
    short vRefNum;
    short *refNum;
OSErr OpenRF(fileName, vRefNum, refNum)
    char *fileName;
    short vRefNum;
    short *refNum;
OSErr FSRead(refNum, count, buffPtr)
    short refNum;
    long *count;
    Ptr buffPtr;
OSErr FSWrite(refNum, count, buffPtr)
    short refNum;
    long *count;
    Ptr buffPtr;
OSErr GetFPos(refNum, filePos)
    short refNum;
    long *filePos;
OSErr SetFPos(refNum, posMode, posOff)
    short refNum;
    short posMode;
    long posOff;
OSErr GetEOF(refNum, logEOF)
    short refNum;
    long *logEOF;
OSErr SetEOF(refNum, logEOF)
    short refNum;
    long logEOF;
OSErr Allocate(refNum, count)

```

```

    short refNum;
    long *count;
    OSErr FSClose(refNum)
    short refNum;

/* Creating and Deleting Files */

OSErr Create(fileName,vRefNum,creator,fileType)
    char *fileName;
    short vRefNum;
    OSType creator;
    OSType fileType;
OSErr FSDelete(fileName,vRefNum)
    char *fileName;
    short vRefNum;

/* Changing Information About Files */

OSErr GetFInfo(fileName,vRefNum,fndrInfo)
    char *fileName;
    short vRefNum;
    FInfo *fndrInfo;
OSErr SetFInfo(fileName,vRefNum,fndrInfo)
    char *fileName;
    short vRefNum;
    FInfo *fndrInfo;
OSErr SetFLock(fileName,vRefNum)
    char *fileName;
    short vRefNum;
OSErr RstFLock(fileName,vRefNum)
    char *fileName;
    short vRefNum;
OSErr Rename(oldName,vRefNum,newName)
    char *oldName;
    short vRefNum;
    char *newName;

/* --- Low-Level Routines ----- */

/* Initializing the File I/O Queue */

void FInitQueue();
void AddDrive(drvrRefNum,drvNum,qEl);
    short drvrRefNum,drvNum;
    DrvQElPtr qEl;
OSErr PBMountVol(paramBlock)
    ParamBlockRec *paramBlock;

/* Accessing Volumes */

OSErr PBGetVInfo(paramBlock,async)
    ParamBlockRec *paramBlock;

```

```

        Boolean async;
OSErr PBHGetVInfo(paramBlock, async)
    HParamBlockRec *paramBlock;
    Boolean async;
OSErr PBSetVInfo(paramBlock, async)
    HParamBlockRec *paramBlock;
    Boolean async;
OSErr PBGetVol(paramBlock, async)
    WDPBPtr paramBlock;
    Boolean async;
OSErr PBHGetVol(paramBlock, async)
    HParamBlockRec *paramBlock;
    Boolean async;
OSErr PBSetVol(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBHSetVol(paramBlock, async)
    WDPBPtr paramBlock;
    Boolean async;
OSErr PBFlushVol(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBUnmountVol(paramBlock)
    ParamBlockRec *paramBlock;
OSErr PBOffLine(paramBlock)
    ParamBlockRec *paramBlock;
OSErr PBEject(paramBlock)
    ParamBlockRec *paramBlock;

/* Accessing Files */

OSErr PBOpen(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBHOpen(paramBlock, async)
    HParamBlockRec *paramBlock;
    Boolean async;
OSErr PBOpenRF(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBHOpenRF(paramBlock, async)
    HParamBlockRec *paramBlock;
    Boolean async;
OSErr PBLockRange(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBUnlockRange(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBRead(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBWrite(paramBlock, async)
    ParamBlockRec *paramBlock;

```

```

    Boolean async;
    OSErr PBGetFPos (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBSetFPos (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBGetEOF (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBSetEOF (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBAllocate (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBAllocContig (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBFlushFile (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBClose (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;

/* Creating and Deleting Files and Directories */

    OSErr PBCreate (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBHCreate (paramBlock, async)
        HParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBDirCreate (paramBlock, async)
        HParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBDelete (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBHDelete (paramBlock, async)
        HParamBlockRec *paramBlock;
    Boolean async;

/* Changing Information About Files and Directories */

    OSErr PBGetFInfo (paramBlock, async)
        ParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBHGetFInfo (paramBlock, async)
        HParamBlockRec *paramBlock;
    Boolean async;
    OSErr PBSetFInfo (paramBlock, async)

```

```

    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBHSetFInfo(paramBlock, async)
    HParamBlockRec *paramBlock;
    Boolean async;
OSErr PBSetFLock(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBHSetFLock(paramBlock, async)
    HParamBlockRec *paramBlock;
    Boolean async;
OSErr PBRstFLock(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBHRstFLock(paramBlock, async)
    HParamBlockRec *paramBlock;
    Boolean async;
OSErr PBSetFVers(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBRename(paramBlock, async)
    ParamBlockRec *paramBlock;
    Boolean async;
OSErr PBHRename(paramBlock, async)
    HParamBlockRec *paramBlock;
    Boolean async;

/* Hierarchical-Only Routines */

OSErr PBGetCatInfo(paramBlock, async)
    CInfoPBRec *paramBlock;
    Boolean async;
OSErr PBSetCatInfo(paramBlock, async)
    CInfoPBRec *paramBlock;
    Boolean async;
OSErr PBCatMove(paramBlock, async)
    CMovePBPtr *paramBlock;
    Boolean async;
OSErr PBOpenWD(paramBlock, async)
    WDPBPtr *paramBlock;
    Boolean async;
OSErr PBCloseWD(paramBlock, async)
    WDPBPtr *paramBlock;
    Boolean async;
OSErr PBGetWDInfo(paramBlock, async)
    WDPBPtr *paramBlock;
    Boolean async;

/* Advanced Routines */

struct QHdr *GetFSQHdr();
struct QHdr *GetVCBQHdr();
struct QHdr *GetDrvQHdr();

```

```
OSErr PBGetFCBInfo(paramBlock, async)
    FCBPBlock *paramBlock;
    Boolean async;
```

Description The File Manager controls the exchange of information between an application and files.

For more detailed information, see the File Manager chapter of *Inside Macintosh*.

Notes Because the global data pointer (register A5) may not be valid at the time an I/O completion routine is executing, that routine cannot safely access any global variables or strings, nor can it call functions outside its segment.

Because C does not have variant records like Pascal, some Pascal records in *Inside Macintosh* are represented by more than one C typedef in this interface.

Warning The low-level routines that use strings take as input and return as output pointers to Pascal-style strings (string length in first byte). However, the high-level routines use C-style strings (terminated by a null character) as input and output parameters.

FixMath—fixed-point math

Synopsis

```
#include <Types.h>
#include <FixMath.h>

/* Routines Available in RAM Library for 64K ROM Users */
/* and in 128K ROM */

/* Arithmetic Operations */

pascal Fixed FixDiv(x,y)
    Fixed x,y;
pascal Fract FracDiv(x,y)
    Fract x,y;
pascal Fract FracMul(x,y)
    Fract x,y;
pascal Fract FracSqrt(x)
    Fract x;
pascal Fract FracSin(x)
    Fixed x;
pascal Fract FracCos(x)
    Fixed x;

/* Routines Available Only in New Macintosh Plus (128K) ROM */

pascal Fixed FixATan2(x,y)
    long x,y;

/* Conversion Functions */

pascal Fixed Long2Fix(x)
    long x;
pascal long Fix2Long(x)
    Fixed x;
pascal Fract Fix2Frac(x)
    Fixed x;
pascal Fixed Frac2Fix(x)
    Fract x;
pascal extended Fix2X(x)
    Fixed x;
Fixed X2Fix(x)
    extended x;
pascal extended Frac2X(x)
    Fract x;
Fract X2Frac(x)
    extended x;
```

Description The FixMath library provides new fixed-point arithmetic routines in addition to those provided by the ToolUtils library.

For more detailed information about the new fixed-point math routines, see the Toolbox Utilities chapter of *Inside Macintosh*, Volume 4.

Fonts—Font Manager

Synopsis

```
#include <Types.h>
#include <Fonts.h>

/* Font Numbers */

#define systemFont      0
#define applFont        1
#define newYork         2
#define geneva          3
#define monaco          4
#define venice          5
#define london          6
#define athens          7
#define sanFran        8
#define toronto         9
#define cairo           11
#define losAngeles      12
#define times           20
#define helvetica       21
#define courier         22
#define symbol          23
#define mobile          24

/* Special Characters */

#define commandMark     '\021'
#define checkMark       '\022'
#define diamondMark     '\023'
#define appleMark       '\024'

/* Font Types */

#define propFont        0x9000
#define prpFntH         0x9001
#define prpFntW         0x9002
#define prpFntHW        0x9003
#define fixedFont       0xB000
#define fxdFntH         0xB001
#define fxdFntW         0xB002
#define fxdFntHW        0xB003
#define fontWid         0xACB0

typedef struct FMInput {
    short          family;      /* for example, New York */
    short          size;       /* for example, 12 Point */
    Style          face;       /* for example, bold or underline */
    Boolean        needBits;   /* bits or just measurement */
}
```

```

    short          device;          /* always 0 for display */
    Point          numer;           /* current drawing scale */
    Point          denom;           /* current drawing scale */
} FMInput;

typedef struct FMOOutput {
    short          errNum;          /* not used */
    Handle         fontHandle;      /* Handle to font */
    unsigned char  boldPixels;      /* pixels of horizontal smear */
    unsigned char  italicPixels;    /* pixels of horizontal smear */
    unsigned char  ulOffset;        /* pixels below baseline */
    unsigned char  ulShadow;        /* pixels in halo */
    unsigned char  ulThick;         /* thickness of underline */
    unsigned char  shadowPixels;    /* pixels to shadow {0..3} */
    char          extra;            /* extra white pixels/char */
    unsigned char  ascent;          /* ascent */
    unsigned char  descent;         /* descent */
    unsigned char  widMax;          /* maximum character width */
    char          leading;          /* leading between lines */
    char          unused;           /* not used */
    Point          numer;           /* current drawing scale */
    Point          denom;           /* current drawing scale */
} FMOOutput, *FMOOutPtr;

typedef struct FontRec {
    short          fontType;        /* font type */
    short          firstChar;       /* ASCII code of first character */
    short          lastChar;        /* ASCII code of last character */
    short          widMax;          /* maximum character width */
    short          kernMax;         /* negative of maximum character kern */
    short          nDescent;        /* negative of descent */
    short          fRectWidth;      /* width of font rectangle */
    short          fRectHeight;     /* height of font rectangle */
    short          owTLoc;          /* offset to offset/width table */
    short          ascent;          /* ascent */
    short          descent;         /* descent */
    short          leading;         /* leading */
    short          rowWords;        /* row width of bit image / 2 */
    short          bitImage[(rowWords-1)+1][(fRectHeight-1)+1];
    short          locTable[(lastChar+2-firstChar)+1];
    short          owTable[(lastChar+2-firstChar)+1];
    short          widthTab[(lastChar+2-firstChar)+1];
    short          heightTab[(lastChar+2-firstChar)+1];
} FontRec;

typedef struct FMetricRec {
    Fixed          ascent;
    Fixed          descent;
    Fixed          leading;
    Fixed          widMax;
    Handle         wTabHandle;
} FMetricRec;

```

```

typedef struct WidthTable {
    Fixed          tabData[256]; /* character widths */
    Handle         tabFont;      /* font record used to build table */
    long           sExtra;       /* space extra used for table */
    long           style;        /* extra because of style */
    short          fID;          /* font family ID */
    short          fSize;        /* font size request */
    short          face;         /* style (face) request */
    short          device;       /* device requested */
    Point          inNumer;      /* vertical input scale factor */
    Point          inDenom;      /* horizontal input scale factor */
    short          aFID;         /* actual font family ID for table */
    Handle         fHand;        /* family record used to build table */
    Boolean        usedFam;      /* used fixed-point family widths? */
    unsigned char  aFace;        /* actual face produced */
    short          vOutput;      /* vert output scale factor--true */
    short          hOutput;      /* horiz output scale factor--true */
    short          vFactor;      /* vert output scale factor--pretty */
    short          hFactor;      /* horiz output scale factor--pretty */
    short          aSize;        /* actual size of actual font used */
    short          tabSize;      /* total size of table */
} WidthTable;

typedef struct FamRec {
    short          ffFlags;      /* flags for family */
    short          ffFamID;      /* family ID number */
    short          ffFirstChar;  /* ASCII code of first char */
    short          ffLastChar;   /* code of last char */
    short          ffAscent;     /* max ascent for 1-pt. font */
    short          ffDescent;    /* max descent for 1-pt. font */
    short          ffLeading;     /* max leading for 1-pt. font */
    short          ffWidMax;     /* max width for 1-pt. font */
    long           ffWTabOff;    /* offset to width table */
    long           ffKernOff;    /* offset to kerning table */
    long           ffStylOff;    /* offset to style-mapping table */
    short          ffProperty[9]; /* style property info */
    short          ffIntl[2];    /* for international use */
    short          ffVersion;    /* version number */
    FontAssoc      ffAssoc;
    WidTable       ffWidthTab;
    StyleTable     ffSyTab;
    KernTable      ffKernTab;
} /*
FamRec;

/* Initializing the Font Manager */

pascal void InitFonts();

/* Getting Font Information */

void GetFontName(fontNum,theName)
    short fontNum;

```

```

    char *theName;
void GetFNum(fontName,theNum)
    char *fontName;
    short *theNum;
pascal Boolean RealFont(fontNum,size)
    short fontNum;
    short size;

/* Keeping Fonts in Memory */

pascal void SetFontLock(lockFlag)
    Boolean lockFlag;

/* Advanced Routines */

pascal FMOutPtr FMSwapFont(inRec)
    FMInput *inRec;
pascal void FontMetrics(theMetrics)
    FMetricRec *theMetrics;
pascal void SetFScaleDisable(fscaleDisable)
    Boolean fscaleDisable;
void SetFractEnable(fractEnable)
    Boolean fractEnable;

```

Description

The Font Manager supports the character fonts used to draw text with QuickDraw. For more detailed information, see the Font Manager chapter of *Inside Macintosh*.

Graf3D—three-dimensional graphics routines

Synopsis

```
#include <Types.h>
#include <QuickDraw.h>
#include <Graf3D.h>

#define radConst 3754936 /* radConst = 57.29578 */

typedef struct Point3D {
    Fixed x,y,z;
} Point3D;

typedef struct Point2D {
    Fixed x,y;
} Point2D;

typedef Fixed XfMatrix[4][4];

typedef struct Port3D {
    GrafPtr grPort;
    Rect viewRect;
    Fixed xLeft,yTop,xRight,yBottom;
    Point3D pen,penPrime,eye;
    Fixed hSize,vSize;
    Fixed hCenter,vCenter;
    Fixed xCotan,yCotan;
    char ident;
    char filler;
    XfMatrix xForm;
} Port3D,*Port3DPtr;

pascal void InitGrf3D(port)
    Port3DPtr *port;
pascal void Open3DPort(port)
    Port3DPtr port;
pascal void SetPort3D(port)
    Port3DPtr port;
pascal void GetPort3D(port)
    Port3D *port;
pascal void MoveTo2D(x,y)
    Fixed x,y;
pascal void MoveTo3D(x,y,z)
    Fixed x,y,z;
pascal void LineTo2D(x,y)
    Fixed x,y;
pascal void LineTo3D(x,y,z)
    Fixed x,y,z;
pascal void Move2D(x,y)
    Fixed x,y;
pascal void Move3D(x,y,z)
    Fixed x,y,z;
```

```

pascal void Line2D(x,y)
    Fixed x,y;
pascal void Line3D(x,y,z)
    Fixed x,y,z;
pascal void ViewPort(r)
    Rect *r;
pascal void LookAt(left,top,right,bottom)
    Fixed left,top,right,bottom;
pascal void ViewAngle(angle)
    Fixed angle;
pascal void Identity();
pascal void Scale(xFactor,yFactor,zFactor)
    Fixed xFactor,yFactor,zFactor;
pascal void Translate(dx,dy,dz)
    Fixed dx,dy,dz;
pascal void Pitch(xAngle)
    Fixed xAngle;
pascal void Yaw(yAngle)
    Fixed yAngle;
pascal void Roll(zAngle)
    Fixed zAngle;
pascal void Skew(zAngle)
    Fixed zAngle;
pascal void Transform(src,dst)
    Point3D *src,*dst;
pascal short Clip3D(src1,src2,dst1,dst2)
    Point3D *src1,*src2;
    Point *dst1,*dst2;
pascal void SetPt3D(pt3D,x,y,z)
    Point3D *pt3D;
    Fixed x,y,z;
pascal void SetPt2D(pt2D,x,y)
    Point2D *pt2D;
    Fixed x,y;

```

Description

The Graf3D routines are an extension of QuickDraw that provide three-dimensional graphics.

For more detailed information, see Appendix D, "Graf3D: Three-Dimensional Graphics."

Lists—List Manager Package

Synopsis

```
#include <Types.h>
#include <Lists.h>

/* Masks for Automatic Scrolling */

#define lDoVAutoscroll    2
#define lDoHAutoscroll    1

/* Masks for Selection Flags */

#define lOnlyOne          (-128)
#define lExtendDrag       0x40
#define lNoDisjoint       0x20
#define lNoExtend         0x10
#define lNoRect            0x08
#define lUseSense          0x04
#define lNoNilHilite       0x02

/* Messages to List Definition Procedure */

#define lInitMsg           0
#define lDrawMsg           1
#define lHiliteMsg         2
#define lCloseMsg          3

/* Data Types */

typedef Point Cell;

typedef struct ListRec {
    Rect      rView;
    GrafPtr   port;
    Point     indent;
    Point     cellSize;
    Rect      visible;
    struct ControlRecord **vScroll;
    struct ControlRecord **hScroll;
    char      selFlags;
    char      lActive;
    char      lReserved;
    char      listFlags;
    long      clikTime;
    Point     clikLoc;
    Point     mouseLoc;
    ProcPtr   lClikLoop;
    Cell      lastClick;
    long      refCon;
}
```



```

    Handle          listDefProc;
    Handle          userHandle;
    Rect            dataBounds;
    Handle          cells;
    short           maxIndex;
    short           cellArray[1];
} ListRec, *ListPtr, **ListHandle;

/* Creating and Disposing of Lists */

ListHandle LNew(rView, dataBounds, cSize, theProc, theWindow,
drawIt, hasGrow, scrollHoriz, scrollVert)
    Rect *rView, *dataBounds;
    Point *cSize;
    short theProc;
    struct GrafPort *theWindow;
    Boolean drawIt, hasGrow, scrollHoriz, scrollVert;
pascal void LDispose(lHandle)
    ListHandle lHandle;

/* Adding and Deleting Rows and Columns */

pascal short LAddColumn(count, colNum, lHandle)
    short count, colNum;
    ListHandle lHandle;
pascal short LAddRow(count, rowNum, lHandle)
    short count, rowNum;
    ListHandle lHandle;
pascal void LDelColumn(count, colNum, lHandle)
    short count, colNum;
    ListHandle lHandle;
pascal void LDelRow(count, rowNum, lHandle)
    short count, rowNum;
    ListHandle lHandle;

/* Operations on Cells */

void LAddToCell(dataPtr, dataLen, theCell, lHandle)
    Ptr dataPtr;
    short dataLen;
    Cell *theCell;
    ListHandle lHandle;
void LClrCell(theCell, lHandle)
    Cell *theCell;
    ListHandle lHandle;
void LGetCell(dataPtr, dataLen, theCell, lHandle)
    Ptr dataPtr;
    short *dataLen;
    Cell *theCell;
    ListHandle lHandle;
void LSetCell(dataPtr, dataLen, theCell, lHandle)
    Ptr dataPtr;
    short dataLen;

```

```

    Cell *theCell;
    ListHandle lHandle;
void LCellSize(cSize,lHandle)
    Point *cSize;
    ListHandle lHandle;
pascal Boolean LGetSelect (next,theCell,lHandle)
    Boolean next;
    Cell *theCell;
    ListHandle lHandle;
void LSetSelect(setIt,theCell,lHandle)
    Boolean setIt;
    Cell *theCell;
    ListHandle lHandle;

/* Mouse Location */

Boolean LClick(pt,modifiers,lHandle)
    Point *pt;
    short modifiers;
    ListHandle lHandle;
pascal Cell LLastClick(lHandle)
    ListHandle lHandle;

/* Accessing Cells */

void LFind(offset,len,theCell,lHandle)
    short *offset,*len;
    Cell *theCell;
    ListHandle lHandle;
pascal Boolean LNextCell(hNext,vNext,theCell,lHandle)
    Boolean hNext,vNext;
    Cell *theCell;
    ListHandle lHandle;
void LRect(cellRect,theCell,lHandle)
    Rect *cellRect;
    Cell *theCell;
    ListHandle lHandle;
pascal Boolean LSearch(dataPtr,dataLen,searchProc,theCell,lHandle)
    Ptr dataPtr;
    short dataLen;
    ProcPtr searchProc;
    Cell *theCell;
    ListHandle lHandle;
pascal void LSize(listWidth,listHeight,lHandle)
    short listWidth,listHeight;
    ListHandle lHandle;

/* List Display */

void LDraw(theCell,lHandle)
    Cell *theCell;
    ListHandle lHandle;
pascal void LDraw(drawIt,lHandle)

```

```

        Boolean drawIt;
        ListHandle lHandle;
pascal void LScroll(dCols,dRows,lHandle)
        short dCols,dRows;
        ListHandle lHandle;
pascal void LAutoScroll(lHandle)
        ListHandle lHandle;
pascal void LUpdate(theRgn,lHandle)
        RgnHandle theRgn;
        ListHandle lHandle;
pascal void LActivate(act,lHandle)
        Boolean act;
        ListHandle lHandle;

```

User routines

```
pascal void
```

```

MyListDef(lMessage,lSelect,lRect,lCell,lDataOffset,
        lDataLen,lHandle)
        short lMessage;
        Boolean lSelect;
        Rect *lRect;
        Cell lCell;
        short lDataOffset,lDataLen;
        ListHandle lHandle;

```

Description

The List Manager Package lets you create, display, and manipulate lists. For more detailed information, see the List Manager Package chapter of *Inside Macintosh*, Volume 4.

Memory—Memory Manager

Synopsis

```
#include <Types.h>
#include <Memory.h>

typedef long Size;

typedef struct Zone {
    Ptr    bkLim;        /* limit pointer */
    Ptr    purgePtr;     /* used internally */
    Ptr    hFstFree;     /* first free master pointer */
    long   zcbFree;      /* number of free bytes */
    ProcPtr gzProc;      /* grow zone function */
    short  moreMast;     /* master pointers to allocate */
    short  flags;        /* used internally */
    short  cntRel;       /* relocatable blocks */
    short  maxRel;       /* maximum cntRel value */
    short  cntNRel;      /* nonrelocatable blocks */
    short  maxNRel;      /* maximum maxRel value */
    short  cntEmpty;     /* empty master pointers */
    short  cntHandles;   /* total master pointers */
    long   minCBFree;    /* minimum zcbFree value */
    ProcPtr purgeProc;   /* purge warning procedure */
    Ptr    sparePtr;     /* used internally */
    Ptr    allocPtr;     /* used internally */
    short  heapData;     /* first usable byte in zone */
} Zone, *THz;

/* Initialization and Allocation */

void InitApplZone();
void SetApplBase(startPtr)
    Ptr startPtr;
void InitZone(pGrowZone, cMoreMasters, limitPtr, startPtr)
    ProcPtr pGrowZone;
    short cMoreMasters;
    Ptr limitPtr, startPtr;
Ptr GetApplLimit();
void SetApplLimit(zoneLimit)
    Ptr zoneLimit;
void MaxApplZone();
void MoreMasters();

/* Heap Zone Access */

THz GetZone();
void SetZone(hz)
    THz hz;
THz SystemZone();
THz ApplicZone();
```

```

/* Allocating and Releasing Relocatable Blocks */

Handle NewHandle(logicalSize)
    Size logicalSize;
void DisposHandle(h)
    Handle h;
Size GetHandleSize(h)
    Handle h;
void SetHandleSize(h,newSize)
    Handle h;
    Size newSize;
THz HandleZone(h)
    Handle h;
Handle RecoverHandle(p)
    Ptr p;
void ReallocHandle(h,logicalSize)
    Handle h;
    Size logicalSize;

/* Allocating and Releasing Nonrelocatable Blocks */

Ptr NewPtr(logicalSize)
    Size logicalSize;
void DisposPtr(p)
    Ptr p;
Size GetPtrSize(p)
    Ptr p;
void SetPtrSize(p,newSize)
    Ptr p;
    Size newSize;
THz PtrZone(p)
    Ptr p;

/* Freeing Space in the Heap */

long FreeMem();
Size MaxMem(grow)
    Size *grow;
Size CompactMem(cbNeeded)
    Size cbNeeded;
void ResrvMem(cbNeeded)
    Size cbNeeded;
void PurgeMem(cbNeeded)
    Size cbNeeded;
void EmptyHandle(h)
    Handle h;

/* Properties of Relocatable Blocks */

void HLock(h)
    Handle h;
void HUnlock(h)
    Handle h;

```

```

void HPurge(h)
    Handle h;
void HNoPurge(h)
    Handle h;

/* Grow Zone Operations */

void SetGrowZone(growZone)
    ProcPtr growZone;
Handle GZSaveHnd();

/* Miscellaneous Routines */

BlockMove(sourcePtr,destPtr,byteCount)
    Ptr sourcePtr, destPtr;
    Size byteCount;
Ptr TopMem();
void MoveHHI(h)
    Handle h;
OSErr MemError();
    pascal long MaxBlock();
void PurgeSpace(total,contig)
    long *total, *contig;
pascal long StackSpace();
pascal Handle NewEmptyHandle();
    void HSetRBit(h)
        Handle h;
void HClrRBit(h)
    Handle h;
int HGetState(h)
    Handle h;
void HSetState(h,flags)
    Handle h;
    short flags;

```

User routines pascal long MyGrowZone(cbNeeded)
 Size cbNeeded;

Description The Memory Manager provides dynamic allocation of both relocatable and nonrelocatable memory space within the system and application heaps.
 For more detailed information, see the Memory Manager chapter of *Inside Macintosh*.

Menus—Menu Manager

Synopsis

```
#include <Types.h>
#include <Menus.h>

/* Special Characters */

#define noMark    '\0'

/* Messages to Menu Definition Functions */

#define mDrawMsg    0
#define mChooseMsg  1
#define mSizeMsg    2

/* Resource ID of Standard Menu Definition Procedure */

#define textMenuProc 0

/* Low Memory Global: Current Text Justification */

#define TESysJust  (*((short *)0xbac))

typedef struct MenuInfo {
    short menuID;
    short menuWidth;
    short menuHeight;
    ProcHandle menuProc;
    long enableFlags;
    Str255 menuData;
} MenuInfo, *MenuPtr, **MenuHandle;

/* Initialization and Allocation */

pascal void InitMenus();
MenuHandle NewMenu(menuID, menuTitle)
    short menuID;
    char *menuTitle;
pascal MenuHandle GetMenu(resourceID)
    short resourceID;
pascal void DisposeMenu(theMenu)
    MenuHandle theMenu;
void AppendMenu(theMenu, data)
    MenuHandle theMenu;
    char *data;
pascal void AddResMenu(theMenu, theType)
    MenuHandle theMenu;
    ResType theType;
pascal void InsertResMenu(theMenu, theType, afterItem)
```

```

MenuHandle      theMenu;
ResType         theType;
short           afterItem;

/* Forming the Menu Bar */

pascal void InsertMenu(theMenu,beforeID)
    MenuHandle theMenu;
    short beforeID;
pascal void DrawMenuBar();
pascal void DeleteMenu(menuID)
    short menuID;
pascal void ClearMenuBar();
pascal Handle GetNewMBar(menuBarID)
    short menuBarID;
pascal Handle GetMenuBar();
pascal void SetMenuBar(menuList)
    Handle menuList;
pascal void DelMenuItem(theMenu,item)
    MenuHandle theMenu;
    short item;
void InsMenuItem(theMenu,itemString,afterItem)
    MenuHandle theMenu;
    char *itemString;
    short afterItem;

/* Choosing From a Menu */

long MenuSelect(startPt)
    Point *startPt;
pascal long MenuKey(ch)
    short ch;
pascal void HiliteMenu(menuID)
    short menuID;

/* Controlling Items' Appearance */

void SetItem(theMenu,item,itemString)
    MenuHandle theMenu;
    short item;
    char *itemString;
void GetItem(theMenu,item,itemString)
    MenuHandle theMenu;
    short item;
    char *itemString;
pascal void DisableItem(theMenu,item)
    MenuHandle theMenu;
    short item;
pascal void EnableItem(theMenu,item)
    MenuHandle theMenu;
    short item;
pascal void CheckItem(theMenu,item,checked)
    MenuHandle theMenu;

```



```

    short item;
    Boolean checked;
pascal void SetItemMark(theMenu,item,markChar)
    MenuHandle theMenu;
    short item;
    short markChar;
pascal void GetItemMark(menu,item,markChar)
    MenuHandle menu;
    short item;
    short *markChar;
pascal void SetItemIcon(theMenu,item,iconNum)
    MenuHandle theMenu;
    short item;
    short iconNum;
pascal void GetItemIcon(theMenu,item,iconNum)
    MenuHandle theMenu;
    short item;
    short *iconNum;
pascal void SetItemStyle(theMenu,item,chStyle)
    MenuHandle theMenu;
    short item;
    Style chStyle;
pascal void GetItemStyle(menu,item,chStyle)
    MenuHandle menu;
    short item;
    Style *chStyle;

/* Miscellaneous Utilities */

pascal void CalcMenuSize(theMenu)
    MenuHandle theMenu;
pascal short CountMItems(theMenu)
    MenuHandle theMenu;
pascal MenuHandle GetMHandle(menuID)
    short menuID;
pascal void FlashMenuBar(menuID)
    short menuID;
pascal void SetMenuFlash(count)
    short count;

```

| | |
|-----------------|---|
| - User routines | <pre> pascal void MyMenu(message,theMenu,menuRect,hitPt,whichItem) short message; MenuHandle theMenu; Rect *menuRect; Point hitPt; short *whichItem; </pre> |
|-----------------|---|

| | |
|-------------|--|
| Description | <p>The Menu Manager provides routines for creating and using menus.</p> <p>For more detailed information, see the Menu Manager chapter of <i>Inside Macintosh</i>.</p> |
|-------------|--|

Warning

The names of desk accessories start with a null byte. The output parameter from `GetMenuItem` will return a string that begins with a null byte when a desk accessory is selected from the Apple menu. `OpenDeskAcc` skips over the null byte when interpreting its parameter.

OSEvents—Operating System Event Manager

Synopsis

```
#include <Types.h>
#include <OSEvents.h>

/* Event Queue Element Structure */

typedef struct EvQEl {
    struct QElem *qLink;
    short    qType;
    short    evtQWhat;
    long     evtQMessage;
    long     evtQWhen;
    Point    evtQWhere;
    short    evtQModifiers;
} EvQEl;

/* Posting and Removing Events */

OSErr PostEvent(eventCode,eventMsg)
    short eventCode;
    long eventMsg;
OSErr PPostEvent(eventCode,eventMsg,qEl)
    short eventCode;
    long eventMsg;
    EvQEl *qEl;
void FlushEvents(eventMask,stopMask)
    short eventMask;
    short stopMask;

/* Accessing Events */

Boolean GetOSEvent(eventMask,theEvent)
    short eventMask;
    struct EventRecord *theEvent;
Boolean OSEventAvail(eventMask,theEvent)
    short eventMask;
    struct EventRecord *theEvent;

/* Setting the System Event Mask */

void SetEventMask(theMask)
    short theMask;

/* Directly Accessing the Event Queue */

struct QHdr *GetEvQHdr();
```

Description The Operating System Event Manager provides a low-level interface to the Macintosh keyboard, keypad, and mouse.

For more detailed information, see the Operating System Event Manager chapter of *Inside Macintosh*.

OSUtils—Operating System Utilities

Synopsis

```
#include <Types.h>
#include <OSUtils.h>

/* Serial Port Configuration Constants for Config Field of SysParmType */

#define useFree 0 /* Use undefined */
#define useATalk 1 /* AppleTalk */
#define useAsync 2 /* Async */

/* Values Returned by Environs Procedure */

#define macXLMachine 0 /* Macintosh XL */
#define macMachine 1 /* Macintosh */

/* Data Types */

/* typedef long OSType; appears in file Types.h */

/* typedef short OSErr; appears in file Types.h */

typedef struct SysParmType {
    char valid; /* validity status */
    char aTalkA; /* AppleTalk node # hint for port A */
    char aTalkB; /* AppleTalk node # hint for port B */
    char config; /* AppleTalk serial port configuration */
                /* port A = bits 4-7, B = bits 0-3 */
    short portA; /* modem port configuration */
    short portB; /* printer port configuration */
    long alarm; /* alarm setting */
    short font; /* default application font number--1 */
    short kbdPrint; /* auto-key threshold and rate, */
                /* printer connection */
    short volClick; /* speaker volume, double-click, and */
                /* caret-blink times */
    short misc; /* mouse scaling, system startup disk, */
                /* and menu blink */
} SysParmType, *SysPPtr;

typedef struct QElem {
    struct QElem *qLink; /* next queue entry */
    short qType; /* queue type */
    short qData[1]; /* queue type specific data */
} QElem, *QElemPtr;

typedef struct QHdr {
    short qFlags; /* queue flags */

```

```

    QElemPtr  qHead; /* first queue entry */
    QElemPtr  qTail; /* last queue entry */
} QHdr, *QHdrPtr;

typedef enum {
    dummyType,
    vType,      /* vertical retrace queue type */
    ioQType,    /* file I/O or driver I/O type */
    drvQType,   /* drive queue type */
    evType,     /* event queue type */
    fsQType     /* volume-control-block queue type */
} QTypes;

typedef struct DateTimeRec {
    short  year;      /* four-digit year */
    short  month;     /* 1 to 12 for January to December */
    short  day;       /* 1 to 31 */
    short  hour;      /* 0 to 23 */
    short  minute;    /* 0 to 59 */
    short  second;    /* 0 to 59 */
    short  dayOfWeek; /* 1 to 7 for Sunday to Saturday */
} DateTimeRec;

typedef enum {
    OSTrap,
    ToolTrap
} TrapType;

/* Pointer and Handle Manipulation */

OSErr HandToHand(theHndl)
    Handle *theHndl;
OSErr PtrToHand(srcPtr,dstHndl,size)
    Ptr srcPtr;
    Handle *dstHndl;
    long size;
OSErr PtrToXHand(srcPtr,dstHndl,size)
    Ptr srcPtr;
    Handle dstHndl;
    long size;
OSErr HandAndHand(aHndl,bHndl)
    Handle aHndl,bHndl;
OSErr PtrAndHand(pntr,hndl,size)
    Ptr pntr;
    Handle hndl;
    long size;

/* String Comparison */

Boolean EqualString(aStr,bStr,caseSens,diacSens)
    char *aStr,*bStr;
    Boolean caseSens,diacSens;
void UpString(theString,diacSens)

```

```

    char *theString;
    Boolean diacSens;
short RelString(aStr,bStr,caseSens,diacSens)
    char *aStr,*bStr;
    Boolean caseSens,diacSens;

/* Date and Time Operations */

OSErr ReadDateTime(secs)
    long *secs;
void GetDateTime(secs)
    long *secs;
OSErr SetDateTime(secs)
    long secs;
void Date2Secs(date,secs)
    DateTimeRec *date;
    long *secs;
void Secs2Date(secs,date)
    long secs;
    DateTimeRec *date;
void GetTime(date)
    DateTimeRec *date;
void SetTime(date)
    DateTimeRec *date;

/* Parameter RAM Operations */

OSErr InitUtil();
SysPPtr GetSysPPtr();
OSErr WriteParam();

/* Queue Manipulation */

void Enqueue(qEntry,theQueue)
    QElemPtr qEntry;
    QHdrPtr theQueue;
OSErr Dequeue(qEntry,theQueue)
    QElemPtr qEntry;
    QHdrPtr theQueue;

/* Dispatch Table Utilities */

void SetTrapAddress(trapAddr,trapNum)
    long trapAddr;
    short trapNum;
long GetTrapAddress(trapNum)
    short trapNum;
long NGetTrapAddress(trapNum,tType)
    short trapNum;
    TrapType tType;
void NSetTrapAddress(trapAddr,trapNum,tType)
    long trapAddr;
    short trapNum;

```

```

    TrapType tType;

/* Miscellaneous Utilities */

void Delay(numTicks,finalTicks)
    long numTicks;
    long *finalTicks;
pascal void SysBeep(duration)
    short duration;
void Environs(rom,machine)
    short *rom,*machine;
void Restart();

```

Description

The Operating System Utilities are a set of routines and data types in the operating system that perform generally useful operations such as manipulating pointers and handles, comparing strings, and reading the date and time.

For more detailed information, see the Operating System Utilities chapter of *Inside Macintosh*.

Packages—Package Manager, Disk Initialization, Standard File Package, International Utilities, Binary-Decimal Conversion

Synopsis

```
#include <Types.h>
#include <Packages.h>

/* Package Manager ----- */
/* Resource IDs for Packages */

#define listMgr 0 /* List Manager */
#define dskInit 2 /* Disk Initialization */
#define stdFile 3 /* Standard File */
#define flPoint 4 /* Floating-Point Arithmetic */
#define trFunct 5 /* Transcendental Functions */
#define intUtil 6 /* International Utilities */
#define bdConv 7 /* Binary-Decimal Conversion */

pascal void InitAllPacks();

pascal void InitPack(packID)
    short packID;

/* Standard File Package ----- */

/* SFPutFile Dialog Template ID */
#define putDlgID (-3999) /* SFPutFile dialog template ID */

/* Item Numbers of Enabled Items in SFPutFile Dialog */

#define putSave 1 /* Save button */
#define putCancel 2 /* Cancel button */
#define putEject 5 /* Eject button */
#define putDrive 6 /* Drive button */
#define putName 7 /* editText item for file name */

/* SFGetFile Dialog Template ID */

#define getDlgID (-4000) /* SFGetFile dialog template ID */

/* Item Numbers of Enabled Items in SFGetFile Dialog */

#define getOpen 1 /* Open button */
#define getCancel 3 /* Cancel button */
#define getEject 5 /* Eject button */
#define getDrive 6 /* Drive button */
#define getNmList 7 /* userItem for filename list */
```

```

#define getScroll 8 /* userItem for scroll bar */

/* Data Types */

typedef struct SFReply {
    Boolean          good;          /* false if ignore command */
    Boolean          copy;          /* not used */
    OSType           fType;         /* file type or not used */
    short            vRefNum;       /* volume reference number */
    short            version;       /* file's version number */
    String(63)       fName;        /* filename */
} SFReply;

typedef OSType SFTYPEList[4];

/* Standard File Operations */

void SFPutFile(where,prompt,origName,dlgHook,reply)
    Point *where;
    char *prompt;
    char *origName;
    ProcPtr dlgHook;
    SFReply *reply;
void SFPPutFile(where,prompt,origName,dlgHook,reply,dlgID,filterProc)
    Point *where;
    char *prompt;
    char *origName;
    ProcPtr dlgHook;
    SFReply *reply;
    short dlgID;
    ProcPtr filterProc;
void SFGetFile(where,prompt,fileFilter,numTypes,typeList,dlgHook,reply)
    Point *where;
    char *prompt;
    ProcPtr fileFilter;
    short numTypes;
    SFTYPEList typeList;
    ProcPtr dlgHook;
    SFReply *reply;
void SFPGetFile(where,prompt,fileFilter,numTypes,typeList,dlgHook,reply,
    dlgID,filterProc)
    Point *where;
    char *prompt;
    ProcPtr fileFilter;
    short numTypes;
    SFTYPEList typeList;
    ProcPtr dlgHook;
    SFReply *reply;
    short dlgID;
    ProcPtr filterProc;

```

```

/* International Utilities Package ----- */

/* Masks for Currency Format */

#define currSymLead 16 /* set if currency symbol leads */
#define currNegSym 32 /* set if minus sign for negative */
#define currTrailingZ 64 /* set if trailing decimal zeros */
#define currLeadingZ 128 /* set if leading integer zeros */

/* Order of Short Date Elements */

#define mdy 0 /* month day year */
#define dmy 1 /* day month year */
#define ymd 2 /* year month day */

/* Masks for Short Date Format */

#define dayLdingZ 32 /* set if leading zero for day */
#define mntLdingZ 64 /* set if leading zero for month */
#define century 128 /* set if century included */

/* Masks for Time Format */

#define secLeadingZ 32 /* set if leading zero for seconds */
#define minLeadingZ 64 /* set if leading zero for minutes */
#define hrLeadingZ 128 /* set if leading zero for hours */

/* High-Order Byte of Version Information */

#define verUS 0
#define verFrance 1
#define verBritain 2
#define verGermany 3
#define verItaly 4
#define verNetherlands 5
#define verBelgiumLux 6
#define verSweden 7
#define verSpain 8
#define verDenmark 9
#define verPortugal 10
#define verFrCanada 11
#define verNorway 12
#define verIsrael 13
#define verJapan 14
#define verAustralia 15
#define verArabia 16
#define verFinland 17
#define verFrSwiss 18
#define verGrSwiss 19
#define verGreece 20
#define verIceland 21
#define verMalta 22

```

```

#define verCyprus 23
#define verTurkey 24
#define verYugoslavia 25

/* Data Types */

typedef struct Intl0Rec {
    char decimalPt; /* decimal point character */
    char thousSep; /* thousands separator */
    char listSep; /* list separator */
    char currSym1; /* currency symbols (3 bytes long) */
    char currSym2;
    char currSym3;
    unsigned char currFmt; /* currency format */
    unsigned char dateOrder; /* short date order--dmy, ymd, or mdy */
    unsigned char shrtDateFmt; /* short date format */
    char dateSep; /* date separator */
    unsigned char timeCycle; /* 0 if 24-hour cycle, 255 if 1-hour */
    unsigned char timeFmt; /* time format */
    char mornStr[4]; /* trailing string for first 12 hours */
    char eveStr[4]; /* trailing string for last 12 hours */
    char timeSep; /* time separator */
    char time1Suff; /* trailing string for 24-hour cycle */
    char time2Suff;
    char time3Suff;
    char time4Suff;
    char time5Suff;
    char time6Suff;
    char time7Suff;
    char time8Suff;
    unsigned char metricSys; /* 255 for metric, 0 if not */
    short intl0Vers; /* version information--country, vers. */
} Intl0Rec, *Intl0Ptr, **Intl0Hndl;

typedef struct Intl1Rec {
    String(15) days[7]; /* day names */
    String(15) months[12]; /* month names */
    unsigned char suppressDay; /* 0 for day name, 255 for none */
    unsigned char lngDateFmt; /* order of long date elements */
    unsigned char dayLeading0; /* 255 for leading 0 in day number */
    unsigned char abbrLen; /* length for abbreviating names */
    char st0[4]; /* date punctuation */
    char st1[4];
    char st2[4];
    char st3[4];
    char st4[4];
    short intl1Vers; /* version information */
    short localRtn[1]; /* routine for string comparison */
} Intl1Rec, *Intl1Ptr, **Intl1Hndl;

typedef enum {
    shortDate,
    longDate,

```

```

    abbrevDate
} DateForm;

/* Routines */

void IUDateString(dateTime,form,result)
    long dateTime;
    DateForm form;
    char *result;
void IUDatePString(dateTime,form,result,intlParam)
    long dateTime;
    DateForm form;
    char *result;
    Handle intlParam;
void IUTimeString(dateTime,wantSeconds,result)
    long dateTime;
    Boolean wantSeconds;
    char *result;
void IUTimePString(dateTime,wantSeconds,result,intlParam)
    long dateTime;
    Boolean wantSeconds;
    char *result;
    Handle intlParam;
Boolean IUMetric();
Handle IUGetIntl(theID)
    short theID;
void IUSetIntl(refNum,theID,intlParam)
    short refNum;
    short theID;
    Handle intlParam;
short IUCompString(aStr,bStr)
    char *aStr,*bStr;
short IUMagString(aPtr,bPtr,aLen,bLen)
    Ptr aPtr,bPtr;
    short aLen,bLen;
short IUEqualString(aStr,bStr)
    char *aStr,*bStr;
short IUMagIDString(aPtr,bPtr,aLen,bLen)
    Ptr aPtr,bPtr;
    short aLen,bLen;

/* Binary-Decimal Conversion Package ----- */

void NumToString(theNum,theString)
    long theNum;
    char *theString;
void StringToNum(theString,theNum)
    char *theString;
    long *theNum;

```

User routines /* Standard File Package ----- */

```
pascal short MyDlg(item,theDialog)
    short item;
    DialogPtr theDialog;
pascal Boolean MyFileFilter(paramBlock)
    ParmBlkPtr paramBlock;
```

Description The Package Manager provides for the initialization of packages. For more detailed information, see the chapters for Package Manager, Disk Initialization Package, Standard File Package, International Utilities Package, and Binary-Decimal Conversion Package in *Inside Macintosh*.

Printing—Printing Manager

Synopsis

```
#include <Types.h>
#include <QuickDraw.h>
#include <Printing.h>

/* Printing Methods */

#define bDraftLoop          0 /* draft printing */
#define bSpoolLoop          1 /* spooling */

/* Printer Specification in prStl Field of Print Record */

#define bDevCItoh           1 /* ImageWriter printer */
#define bDevLaser           3 /* LaserWriter printer */

/* Maximum Number of Pages in a Spool File */

#define iPFMaxPgs           128 /* max pages in a spool file */
#define iPrPgFract          120 /* paper units per inch */

/* Result Codes */

#define iPrSavPFil          (-1) /* saving spool file */
#define iIOAbort            (-27) /* I/O abort error */
#define iPrAbort            128 /* application- or user-requested abort */

/* PrCtlCall Parameters */

#define iPrDevCtl            7 /* device control */
#define lPrReset             0x00010000 /* reset printer */
#define lPrLineFeed          0x00030000 /* start new line */
#define lPrLFSixth           0x0003FFFF /* standard 1/6" line feed */
#define lPrPageEnd           0x00020000 /* start new page */
#define iPrBitsCtl           4 /* bit-map printing */
#define lScreenBits          0 /* configurable */
#define lPaintBits           1 /* 72 x 72 dots */
#define iPrIOCtl            5 /* text streaming */

/* Printer Driver Information */

#define sPrDrvr              ".Print" /* Printer Driver resource name */
#define iPrDrvrRef           (-3) /* Printer Driver reference number */
#define bDevCItoh            1
#define iDevCItoh            (bDevCItoh << 8)
#define bDevDaisy            2
#define iDevDaisy            (bDevDaisy << 8)
#define bDevLaser            3
```

```

#define iDevLaser (bDevLaser << 8)

/* Type Definitions */

typedef Rect *TPRect;

typedef struct TPrPort {
    GrafPort      gPort;          /* graph port to draw in */
    QDProcs       gProcs;         /* pointers to drawing routines */
    long          lGParam1;       /* internal */
    long          lGParam2;       /* internal */
    long          lGParam3;       /* internal */
    long          lGParam4;       /* internal */
    Boolean        fOurPtr;       /* internal */
    Boolean        fOurBits;      /* internal */
} TPrPort, *TPPrPort;

typedef struct TPrInfo {
    short          iDev;           /* printer information */
    short          iVRes;          /* printer vertical resolution */
    short          iHRes;          /* printer horizontal resolution */
    Rect           rPage;         /* page rectangle */
} TPrInfo;

typedef enum {
    feedCut,
    feedFanfold,
    feedMechCut,
    feedOther
} TFeed;

typedef struct TPrStl {
    short          wDev;           /* high byte specifies device */
    short          iPageV;         /* paper height */
    short          iPageH;         /* paper width */
    char           bPort;         /* printer or modem port--ignored */
    TFeed          feed;          /* paper type */
} TPrStl;

typedef enum {
    scanTB,
    scanBT,
    scanLR,
    scanRL
} TScan;

typedef struct TPrXInfo {
    short          iRowBytes;      /* bytes per row */
    short          iBandV;         /* vertical dots */
    short          iBandH;         /* horizontal dots */
    short          iDevBytes;      /* size of bit image */
    short          iBands;         /* bands per page */
    char           bPatScale;     /* used by QuickDraw */
}

```



```

char          bULThick;      /* underline thickness */
char          bULOffset;    /* underline offset */
char          bULShadow;    /* underline descender */
TScan        scan;         /* scan direction */
char          bXInfoX;      /* not used */
} TPrXInfo;

typedef struct TPrJob {
short          iFstPage;     /* first page to print */
short          iLstPage;     /* last page to print */
short          iCopies;      /* number of copies */
char          bJDocLoop;    /* printing method */
Boolean        fFromUser;    /* true if called from application */
ProcPtr        pIdleProc;   /* background procedure */
StringPtr      pFileName;   /* spool-file name */
short          iFileVol;     /* spool-file volume reference number */
char          bFileVers;    /* spool-file version number */
char          bJobX;        /* not used */
} TPrJob;

typedef struct TPrint {
short          iPrVersion;   /* Printing Manager version */
TPrInfo        prInfo;      /* printing information */
Rect           rPaper;      /* paper rectangle */
TPrStl         prStl;       /* style information */
TPrInfo        prInfoPT;    /* copy of prInfo */
TPrXInfo       prXInfo;     /* band information */
TPrJob         prJob;       /* job information */
short          printX[19]    /* internal */
} TPrint, *TPPrint, **THPrint;

typedef struct TPrStatus {
short          iTotPages;    /* total number of pages */
short          iCurPage;    /* page being printed */
short          iTotCopies;   /* number of copies */
short          iCurCopy;    /* copy begin printed */
short          iTotBands;    /* bands per page */
short          iCurBand;    /* band being printed */
Boolean        fPgDirty;     /* true if started printing page */
Boolean        fImaging;     /* true if imaging */
THPrint        hPrint;       /* print record */
TPPrPort       pPrPort;     /* printing port */
PicHandle      hPic;        /* internal */
} TPrStatus;

/* Initialization and Termination */

pascal void PrOpen();

pascal void PrClose();

```

```

/* Print Records and Dialogs */

pascal void PrintDefault(hPrint)
    THPrint hPrint;
pascal Boolean PrValidate(hPrint)
    THPrint hPrint;
pascal Boolean PrStlDialog(hPrint)
    THPrint hPrint;
pascal Boolean PrJobDialog(hPrint)
    THPrint hPrint;
pascal void PrJobMerge(hPrintSrc,hPrintDst)
    THPrint hPrintSrc,hPrintDst;

/* Printing */

pascal TPrPort PrOpenDoc(hPrint,pPrPort,pIOBuf)
    THPrint hPrint;
    TPrPort pPrPort;
    Ptr pIOBuf;
pascal void PrOpenPage(pPrPort,pPageFrame)
    TPrPort pPrPort;
    TRect pPageFrame;
pascal void PrClosePage(pPrPort)
    TPrPort pPrPort;
pascal void PrCloseDoc(pPrPort)
    TPrPort pPrPort;

/* Spool Printing */

pascal void PrPicFile(hPrint,pPrPort,pIOBuf,pDevBuf,prStatus)
    THPrint hPrint;
    TPrPort pPrPort;
    Ptr pIOBuf;
    Ptr pDevBuf;
    TPrStatus *prStatus;

/* Error Handling */

pascal short PrError();
pascal void PrSetError(iErr)
    short iErr;

/* Low-Level Driver Access */

pascal void PrDrvrOpen();
pascal void PrDrvrClose();
pascal void PrCtlCall(iWhichCtl,lParam1,lParam2,lParam3)
    short iWhichCtl;
    long lParam1,lParam2,lParam3;
pascal Handle PrDrvrDCE();
pascal short PrDrvrVers();

```

Description

The Printing Manager supports printing on a variety of devices.

For more detailed information, see the Printing Manager chapter of *Inside Macintosh*.

QuickDraw—graphics routines

Synopsis

```
/* Source Transfer Modes */

#define srcCopy      0
#define srcOr        1
#define srcXor       2
#define srcBic       3
#define notSrcCopy   4
#define notSrcOr     5
#define notSrcXor    6
#define notSrcBic    7

/* Pattern Transfer Modes */

#define patCopy      8
#define patOr        9
#define patXor      10
#define patBic      11
#define notPatCopy  12
#define notPatOr    13
#define notPatXor   14
#define notPatBic   15

/* QuickDraw Color-Separation Constants */

#define normalBit     0
#define inverseBit    1
#define redBit        4 /* RGB additive mapping */
#define greenBit      3
#define blueBit       2
#define cyanBit       8 /* CMYBk subtractive mapping */
#define magentaBit    7
#define yellowBit     6
#define blackBit      5

/* Standard Colors for ForeColor and BackColor */

#define blackColor    33 /* colors expressed in these mappings */
#define whiteColor    30
#define redColor      205
#define greenColor    341
#define blueColor     409
#define cyanColor     273
#define magentaColor  137
#define yellowColor   69

/* Standard Picture Comments */

#define picLParen      0
#define picRParen      1
```

```

/* Type-Style Constants */

#define normal      0x00
#define bold        0x01
#define italic      0x02
#define underline   0x04
#define outline     0x08
#define shadow      0x10
#define condense    0x20
#define extend      0x40

/* Data Types */

typedef unsigned char Pattern[8];
typedef short Bits16[16];
typedef enum {frame,paint,erase,invert,fill} GrafVerb;

/* The typedefs Style, Point, and Rect appear in file Types.h.*/

typedef struct FontInfo {
    short      ascent;
    short      descent;
    short      widMax;
    short      leading;
} FontInfo;

typedef struct BitMap {
    Ptr        baseAddr;
    short      rowBytes;
    Rect        bounds;
} BitMap;

typedef struct Cursor {
    Bits16      data;
    Bits16      mask;
    Point        hotSpot;
} Cursor;

typedef struct PenState {
    Point        pnLoc;
    Point        pnSize;
    short        pnMode;
    Pattern      pnPat;
} PenState;

typedef struct Region {
    short        rgnSize;
    Rect        rgnBBox;
    short        rgnData[1];
} Region, *RgnPtr, **RgnHandle;

typedef struct Picture {

```

```

        short                picSize;
        Rect                 picFrame;
        short                picData[1];
    } Picture, *PicPtr, **PicHandle;

typedef struct Polygon {
    short                polySize;
    Rect                 polyBBox;
    Point                polyPoints[1];
} Polygon, *PolyPtr, **PolyHandle;

typedef struct QDProcs {
    ProcPtr              textProc;
    ProcPtr              lineProc;
    ProcPtr              rectProc;
    ProcPtr              rRectProc;
    ProcPtr              ovalProc;
    ProcPtr              arcProc;
    ProcPtr              polyProc;
    ProcPtr              rgnProc;
    ProcPtr              bitsProc;
    ProcPtr              commentProc;
    ProcPtr              txMeasProc;
    ProcPtr              getPicProc;
    ProcPtr              putPicProc;
} QDProcs, *QDProcsPtr;

typedef struct GrafPort {
    short                device;
    BitMap               portBits;
    Rect                 portRect;
    RgnHandle            visRgn;
    RgnHandle            clipRgn;
    Pattern              bkPat;
    Pattern              fillPat;
    Point                pnLoc;
    Point                pnSize;
    short               pnMode;
    Pattern              pnPat;
    short               pnVis;
    short               txFont;
    Style                txFace;
    short               txMode;
    short               txSize;
    Fixed                spExtra;
    long                 fgColor;
    long                 bkColor;
    short               colrBit;
    short               patStretch;
    PicHandle            picSave;
    RgnHandle            rgnSave;
    PolyHandle           polySave;
    QDProcsPtr           grafProcs;
} GrafPort, *GrafPtr;

```

```

/* External Variable Declarations */

extern struct qd {
    char          private[78];
    long          randSeed;
    BitMap        screenBits;
    Cursor        arrow;
    Pattern       dkGray;
    Pattern       ltGray;
    Pattern       gray;
    Pattern       black;
    Pattern       white;
    GrafPtr       thePort;
} qd;

/* GrafPort Routines */

pascal void InitGraf(globalPtr)
    Ptr globalPtr;
pascal void OpenPort(port)
    GrafPtr port;
pascal void InitPort(port)
    GrafPtr port;
pascal void ClosePort(port)
    GrafPtr port;
pascal void SetPort(port)
    GrafPtr port;
pascal void GetPort(port)
    GrafPtr *port;
pascal void GrafDevice(device)
    short device;
pascal void SetPortBits(bm)
    BitMap *bm;
pascal void PortSize(width,height)
    short width,height;
pascal void MovePortTo(leftGlobal,rightGlobal)
    short leftGlobal,rightGlobal;
pascal void SetOrigin(h,v)
    short h,v;
pascal void SetClip(rgn)
    RgnHandle rgn;
pascal void GetClip(rgn)
    RgnHandle rgn;
pascal void ClipRect(r)
    Rect *r;
pascal void BackPat(pat)
    Pattern pat;

/* Cursor Handling */

pascal void InitCursor();
pascal void SetCursor(crsr)
    Cursor *crsr;

```

```

pascal void HideCursor();
pascal void ShowCursor();
pascal void ObscureCursor();

/* Pen and Line Drawing */

pascal void HidePen();
pascal void ShowPen();
pascal void GetPen(pt)
    Point *pt;
pascal void GetPenState(pnState)
    PenState *pnState;
pascal void SetPenState(pnState)
    PenState *pnState;
pascal void PenSize(width,height)
    short width,height;
pascal void PenMode(mode)
    short mode;
pascal void PenPat(pat)
    Pattern pat;
pascal void PenNormal();
pascal void MoveTo(h,v)
    short h,v;
pascal void Move(dh,dv)
    short dh,dv;
pascal void LineTo(h,v)
    short h,v;
pascal void Line(dh,dv)
    short dh,dv;

/* Text Drawing */

pascal void TextFont(font)
    short font;
pascal void TextFace(face)
    Style face;
pascal void TextMode(mode)
    short mode;
pascal void TextSize(size)
    short size;
pascal void SpaceExtra(extra)
    Fixed extra;
pascal void DrawChar(ch)
    short ch;
void DrawString(s)
    char *s;
pascal void DrawText(textBuf,firstByte,byteCount)
    Ptr textBuf;
    short firstByte,byteCount;
pascal short CharWidth(ch)
    short ch;
short StringWidth(s)
    char *s;

```



```

pascal short TextWidth(textBuf,firstByte,byteCount)
    Ptr textBuf;
    short firstByte,byteCount;
pascal void MeasureText(count,textAddr,charLocs)
    short count;
    Ptr textAddr,charLocs;
pascal void GetFontInfo(info)
    FontInfo *info;

/* Drawing in Color */

pascal void ForeColor(color)
    long color;
pascal void BackColor(color)
    long color;
pascal void ColorBit(whichBit)
    short whichBit;

/* Calculations With Rectangles */

pascal void SetRect(r,left,top,right,bottom)
    Rect *r;
    short left,top,right,bottom;
pascal void OffsetRect(r,dh,dv)
    Rect *r;
    short dh,dv;
pascal void InsetRect(r,dh,dv)
    Rect *r;
    short dh,dv;
pascal Boolean SectRect(src1,src2,dstRect)
    Rect *src1,*src2;
    Rect *dstRect;
pascal void UnionRect(src1,src2,dstRect)
    Rect *src1,*src2;
    Rect *dstRect;
Boolean PtInRect(pt,r)
    Point *pt;
    Rect *r;
void Pt2Rect(pt1,pt2,dstRect)
    Point *pt1,*pt2;
    Rect *dstRect;
void PtToAngle(r,pt,angle)
    Rect *r;
    Point *pt;
    short *angle;
pascal Boolean EqualRect(rect1,rect2)
    Rect *rect1,*rect2;
pascal Boolean EmptyRect(r)
    Rect *r;

```

```

/* Graphic Operations on Rectangles */

pascal void FrameRect(r)
    Rect *r;
pascal void PaintRect(r)
    Rect *r;
pascal void EraseRect(r)
    Rect *r;
pascal void InvertRect(r)
    Rect *r;
pascal void FillRect(r,pat)
    Rect *r;
    Pattern pat;

/* Graphic Operations on Ovals */

pascal void FrameOval(r)
    Rect *r;
pascal void PaintOval(r)
    Rect *r;
pascal void EraseOval(r)
    Rect *r;
pascal void InvertOval(r)
    Rect *r;
pascal void FillOval(r,pat)
    Rect *r;
    Pattern pat;

/* Graphic Operations on Rounded-Corner Rectangles */

pascal void FrameRoundRect(r,ovalWidth,ovalHeight)
    Rect *r;
    short ovalWidth,ovalHeight;
pascal void PaintRoundRect(r,ovalWidth,ovalHeight)
    Rect *r;
    short ovalWidth,ovalHeight;
pascal void EraseRoundRect(r,ovalWidth,ovalHeight)
    Rect *r;
    short ovalWidth,ovalHeight;
pascal void InvertRoundRect(r,ovalWidth,ovalHeight)
    Rect *r;
    short ovalWidth,ovalHeight;
pascal void FillRoundRect(r,ovalWidth,ovalHeight,pat)
    Rect *r;
    short ovalWidth,ovalHeight;
    Pattern pat;

/* Graphic Operations on Arcs and Wedges */

pascal void FrameArc(r,startAngle,arcAngle)
    Rect *r;
    short startAngle,arcAngle;
pascal void PaintArc(r,startAngle,arcAngle)

```

```

    Rect *r;
    short startAngle,arcAngle;
pascal void EraseArc(r,startAngle,arcAngle)
    Rect *r;
    short startAngle,arcAngle;
pascal void InvertArc(r,startAngle,arcAngle)
    Rect *r;
    short startAngle,arcAngle;
pascal void FillArc(r,startAngle,arcAngle,pat)
    Rect *r;
    short startAngle,arcAngle;
    Pattern pat;

/* Calculations With Regions */

pascal RgnHandle NewRgn();
pascal void OpenRgn();
pascal void CloseRgn(dstRgn)
    RgnHandle dstRgn;
pascal void DisposeRgn(rgn)
    RgnHandle rgn;
pascal void CopyRgn(srcRgn,dstRgn)
    RgnHandle srcRgn,dstRgn;
pascal void SetEmptyRgn(rgn)
    RgnHandle rgn;
pascal void SetRectRgn(rgn,left,top,right,bottom)
    RgnHandle rgn;
    short left,top,right,bottom;
pascal void RectRgn(rgn,r)
    RgnHandle rgn;
    Rect *r;
pascal void OffsetRgn(rgn,dh,dv)
    RgnHandle rgn;
    short dh,dv;
pascal void InsetRgn(rgn,dh,dv)
    RgnHandle rgn;
    short dh,dv;
pascal void SectRgn(srcRgnA,srcRgnB,dstRgn)
    RgnHandle srcRgnA,srcRgnB,dstRgn;
pascal void UnionRgn(srcRgnA,srcRgnB,dstRgn)
    RgnHandle srcRgnA,srcRgnB,dstRgn;
pascal void DiffRgn(srcRgnA,srcRgnB,dstRgn)
    RgnHandle srcRgnA,srcRgnB,dstRgn;
pascal void XorRgn(srcRgnA,srcRgnB,dstRgn)
    RgnHandle srcRgnA,srcRgnB,dstRgn;
Boolean PtInRgn(pt,rgn)
    Point *pt;
    RgnHandle rgn;
pascal Boolean RectInRgn(r,rgn)
    Rect *r;
    RgnHandle rgn;
pascal Boolean EqualRgn(rgnA,rgnB)
    RgnHandle rgnA,rgnB;
pascal Boolean EmptyRgn(rgn)

```

```

    RgnHandle rgn;

/* Graphic Operations on Regions */

pascal void FrameRgn(rgn)
    RgnHandle rgn;
pascal void PaintRgn(rgn)
    RgnHandle rgn;
pascal void EraseRgn(rgn)
    RgnHandle rgn;
pascal void InvertRgn(rgn)
    RgnHandle rgn;
pascal void FillRgn(rgn, pat)
    RgnHandle rgn;
    Pattern pat;

/* Bit Transfer Operations */

pascal void ScrollRect(r, dh, dv, updateRgn)
    Rect *r;
    short dh, dv;
    RgnHandle updateRgn;
pascal void CopyBits(srcBits, dstBits, srcRect, dstRect, mode, maskRgn)
    BitMap *srcBits, *dstBits;
    Rect *srcRect, *dstRect;
    short mode; RgnHandle maskRgn;
pascal void SeedFill(srcPtr, dstPtr, srcRow, dstRow, height, words, seedH, seedV)
    Ptr srcPtr, dstPtr;
    short srcRow, dstRow, height, words;
    short seedH, seedV;
pascal void CalcMask(srcPtr, dstPtr, srcRow, dstRow, height, words)
    Ptr srcPtr, dstPtr;
    short srcRow, dstRow, height, words;
pascal void CopyMask(srcBits, maskBits, dstBits, srcRect, maskRect, dstRect)
    BitMap *srcBits, *maskBits, *dstBits;
    Rect *srcRect, *maskRect, *dstRect;

/* Pictures */

pascal PicHandle OpenPicture(picFrame)
    Rect *picFrame;
pascal void PicComment(kind, dataSize, dataHandle)
    short kind, dataSize;
    Handle dataHandle;
pascal void ClosePicture();
pascal void DrawPicture(myPicture, dstRect)
    PicHandle myPicture;
    Rect *dstRect;
pascal void KillPicture(myPicture)
    PicHandle myPicture;

```

```

/* Calculations With Polygons */

pascal PolyHandle OpenPoly();
pascal void ClosePoly();
pascal void KillPoly(poly)
    PolyHandle poly;
pascal void OffsetPoly(poly,dh,dv)
    PolyHandle poly;
    short dh,dv;

/* Graphic Operations on Polygons */

pascal void FramePoly(poly)
    PolyHandle poly;
pascal void PaintPoly(poly)
    PolyHandle poly;
pascal void ErasePoly(poly)
    PolyHandle poly;
pascal void InvertPoly(poly)
    PolyHandle poly;
pascal void FillPoly(poly,pat)
    PolyHandle poly;
    Pattern pat;

/* Calculations With Points */

void AddPt(srcPt,dstPt)
    Point *srcPt,*dstPt;
void SubPt(srcPt,dstPt)
    Point *srcPt,*dstPt;
pascal void SetPt(pt,h,v)
    Point *pt;
    short h,v;
Boolean EqualPt(pt1,pt2)
    Point *pt1,*pt2;
pascal void LocalToGlobal(pt)
    Point *pt;
pascal void GlobalToLocal(pt)
    Point *pt;

/* Miscellaneous Routines */

pascal short Random();
pascal Boolean GetPixel(h,v)
    short h,v;
void StuffHex(thingPtr,s)
    Ptr thingPtr;
    char *s;
pascal void ScalePt(pt,srcRect,dstRect)
    Point *pt;
    Rect *srcRect,*dstRect;
pascal void MapPt(pt,srcRect,dstRect)
    Point *pt;

```

```

    Rect *srcRect,*dstRect;
pascal void MapRect(r,srcRect,dstRect)
    Rect *r;
    Rect *srcRect,*dstRect;
pascal void MapRgn(rgn,srcRect,dstRect)
    RgnHandle rgn;
    Rect *srcRect,*dstRect;
pascal void MapPoly(poly,srcRect,dstRect)
    PolyHandle poly;
    Rect *srcRect,*dstRect;

/* Customizing QuickDraw Operations */

pascal void SetStdProcs(procs)
    QDProcs *procs;
void StdText(byteCount,textbuf,numer,denom)
    short byteCount;
    Ptr textbuf;
    Point *numer,*denom;
void StdLine(newPt)
    Point *newPt;
pascal void StdRect(verb,r)
    GrafVerb verb;
    Rect *r;
pascal void StdRRect(verb,r,ovalWidth,ovalHeight)
    GrafVerb verb;
    Rect *r;
    short ovalWidth,ovalHeight;
pascal void StdOval(verb,r)
    GrafVerb verb;
    Rect *r;
pascal void StdArc(verb,r,startAngle,arcAngle)
    GrafVerb verb;
    Rect *r;
    short startAngle,arcAngle;
pascal void StdPoly(verb,poly)
    GrafVerb verb;
    PolyHandle poly;
pascal void StdRgn(verb,rgn)
    GrafVerb verb;
    RgnHandle rgn;
pascal void StdBits(srcBits,srcRect,dstRect,mode,maskRgn)
    BitMap *srcBits;
    Rect *srcRect,*dstRect;
    short mode;
    RgnHandle maskRgn;
pascal void StdComment(kind,dataSize,dataHandle)
    short kind,dataSize;
    Handle dataHandle;
pascal short StdTxMeas(byteCount,textAddr,numer,denom,info)
    short byteCount;
    Ptr textAddr; Point *numer,*denom;
    FontInfo *info;
pascal void StdGetPic(dataPtr,byteCount)

```

```

    Ptr dataPtr;
    short byteCount;
pascal void StdPutPic(dataPtr,byteCount)
    Ptr dataPtr;
    short byteCount;

```

User routines

```

pascal void MyText (byteCount,textbuf,numer,denom)
    short byteCount;
    Ptr textbuf;
    Point numer,denom;
pascal void MyLine(newPt)
    Point newPt;
pascal void MyRect (verb,r)
    GrafVerb verb;
    Rect *r;
pascal void MyRRect (verb,r,ovalWidth,ovalHeight)
    GrafVerb verb;
    Rect *r;
    short ovalWidth,ovalHeight;
pascal void MyOval (verb,r)
    GrafVerb verb;
    Rect *r;
pascal void MyArc (verb,r,startAngle,arcAngle)
    GrafVerb verb;
    Rect *r;
    short startAngle,arcAngle;
pascal void MyPoly (verb,poly)
    GrafVerb verb;
    PolyHandle poly;
pascal void MyRgn (verb,rgn)
    GrafVerb verb;
    RgnHandle rgn;
pascal void MyBits (srcBits,srcRect,dstRect,mode,maskRgn)
    BitMap *srcBits;
    Rect *srcRect,*dstRect;
    short mode;
    RgnHandle maskRgn;
pascal void MyComment (kind,dataSize,dataHandle)
    short kind,dataSize;
    Handle dataHandle;
pascal short MyTxMeas (byteCount,textAddr,numer,denom,info)
    short byteCount;
    Ptr textAddr;
    Point *numer,*denom;
    FontInfo *info;
pascal void MyGetPic (dataPtr,byteCount)
    Ptr dataPtr;
    short byteCount;
pascal void MyPutPic (dataPtr,byteCount)
    Ptr dataPtr;
    short byteCount;

```

- Description** QuickDraw is the Macintosh graphics package.
For more detailed information, see the QuickDraw chapter of *Inside Macintosh*.
- Warning** User routines `MyText` and `MyLine` are not identical to their counterparts `StdText` and `StdLine`. Point parameters to `MyText` and `MyLine` are passed by value; the corresponding parameters to `StdText` and `StdLine` are passed by reference.

Resources—Resource Manager

Synopsis

```
#include <Types.h>
#include <Resources.h>0

/* Masks for Resource Attributes */

#define resSysHeap 64 /* set if read into system heap */
#define resPurgeable 32 /* set if purgeable */
#define resLocked 16 /* set if locked */
#define resProtected 8 /* set if protected */
#define resPreload 4 /* set if to be preloaded */
#define resChanged 2 /* set if written to resource file */

/* Masks for Resource File Attributes */

#define mapReadOnly 128 /* set if file is read-only */
#define mapCompact 64 /* set to compact file on update */
#define mapChanged 32 /* set if write map on update */

/* typedef long ResType; appears in file Types.h */

/* Initialization */

pascal short InitResources();
pascal void RsrcZoneInit();

/* Opening and Closing Resource Files */

void CreateResFile(fileName)
    char *fileName;
short OpenResFile(fileName)
    char *fileName;
pascal void CloseResFile(refNum)
    short refNum;

/* Checking for Errors */

pascal short ResError();

/* Setting the Current Resource File */

pascal short CurResFile();
pascal short HomeResFile(theResource)
    Handle theResource;
pascal void UseResFile(refNum)
    short refNum;
```

```

/* Getting Resource Types */

pascal short CountTypes();
pascal short Count1Types();
pascal void GetIndType(theType,index)
    ResType *theType;
    short index;
pascal void Get1IndType(theType,index)
    ResType *theType;
    short index;

/* Getting and Disposing of Resources */

pascal void SetResLoad(load)
    Boolean load;
pascal short CountResources(theType)
    ResType theType;
pascal short Count1Resources(theType)
    ResType theType;
pascal Handle GetIndResource(theType,index)
    ResType theType;
    short index;
pascal Handle Get1IndResource(theType,index)
    ResType theType;
    short index;
pascal Handle GetResource(theType,theID)
    ResType theType;
    short theID;
pascal Handle Get1Resource(theType,theID)
    ResType theType;
    short theID;
Handle GetNamedResource(theType,name)
    ResType theType;
    char *name;
Handle Get1NamedResource(theType,name)
    ResType theType;
    char *name;
pascal void LoadResource(theResource)
    Handle theResource;
pascal void ReleaseResource(theResource)
    Handle theResource;
pascal void DetachResource(theResource)
    Handle theResource;

/* Getting Resource Information */

pascal short UniqueID(theType)
    ResType theType;
pascal short Unique1ID(theType)
    ResType theType;
void GetResInfo(theResource,theID,theType,name)
    Handle theResource;
    short *theID;

```

```

    ResType *theType;
    char *name;
pascal short GetResAttrs(theResource)
    Handle theResource;
pascal long SizeResource(theResource)
    Handle theResource;
pascal long MaxSizeRsrc(theResource)
    Handle theResource;
pascal long RsrcMapEntry(theResource)
    Handle theResource;

/* Modifying Resources */

void SetResInfo(theResource,theID,name)
    Handle theResource;
    short theID;
    char *name;
pascal void SetResAttrs(theResource,attrs)
    Handle theResource;
    short attrs;
pascal void ChangedResource(theResource)
    Handle theResource;
void AddResource(theData,theType,theID,name)
    Handle theData;
    ResType theType;
    short theID;
    char *name;
pascal void RmveResource(theResource)
    Handle theResource;
pascal void UpdateResFile(refNum)
    short refNum;
pascal void WriteResource(theResource)
    Handle theResource;
pascal void SetResPurge(install)
    Boolean install;

/* Advanced Routines */

pascal short GetResFileAttrs(refNum)
    short refNum;
pascal void SetResFileAttrs(refNum,attrs)
    short refNum;
    short attrs;
short OpenRFPPerm(fileName, vRefNum, permission)
    char *fileName;
    short vRefNum;
    short permission;

```

Description

The Resource Manager provides access to Macintosh resource files. ResType may be specified as a character literal (for example, 'MENU').

For more detailed information, see the Resource Manager chapter of *Inside Macintosh*.

Retrace—Vertical Retrace Manager

Synopsis

```
#include <Types.h>
#include <Retrace.h>

/* Data Types and Routines */

typedef struct VBLTask {
    struct QElem *qLink; /* next queue entry */
    short qType; /* unique id for validity check */
    ProcPtr vblAddr; /* address of service routine */
    short vblCount; /* count field for timeout */
    short vblPhase; /* phase to allow synchronization */
} VBLTask;
OSErr VInstall(VBLTaskPtr)
    struct QElem *vblTaskPtr;
OSErr VRemove(VBLTaskPtr)
    struct QElem *vblTaskPtr;
struct QHdr *GetVBLQHdr();
```

Description

The Vertical Retrace Manager schedules and performs recurrent tasks during vertical-retrace interrupts.

For more detailed information, see the Vertical Retrace Manager chapter of *Inside Macintosh*.

SANE—Standard Apple Numeric Environment routines

Synopsis

```
#include <SANE.h>

/* Decimal Representation Constants */

#define SIGDIGLEN 20 /* significant decimal digits */
#define DECSTROUTLEN 80 /* max length for decimal string */

/* Decimal Formatting Styles */

#define FLOATDECIMAL 0
#define FIXEDDECIMAL 1

/* Exceptions */

#define INVALID 1
#define UNDERFLOW 2
#define OVERFLOW 4
#define DIVBYZERO 8
#define INEXACT 16

/* Ordering Relations */

#define GREATERTHAN 0
#define LESSTHAN 1
#define EQUALTO 2
#define UNORDERED 3

/* Inquiry Classes */

#define SNAN 0
#define QNAN 1
#define INFINITE 2
#define ZERONUM 3
#define NORMALNUM 4
#define DENORMALNUM 5

/* Rounding Directions */

#define TONEAREST 0
#define UPWARD 1
#define DOWNWARD 2
#define TOWARDZERO 3

/* Rounding Precisions */

#define EXTPRECISION 0
```

```

#define DBLPRECISION 1
! #define FLOATPRECISION 2

/* Type Definitions */

typedef short exception; /* sum of INVALID...INEXACT */
typedef short relop; /* relational operator */
typedef short numclass; /* inquiry class */
typedef short rounddir; /* rounding direction */
typedef short roundpre; /* rounding precision */
typedef short environment;
typedef struct decimal {
    char sgn, unused; /* sign 0 for +, 1 for - */
    short exp; /* decimal exponent */
    struct {unsigned char length, text(SIGDIGLEN), unused} sig;
    /* significant digits */
} decimal;

typedef struct decform {
    char style, unused; /* FLOATDECIMAL or FIXEDDECIMAL */
    short digits;
} decform;

typedef void (*haltvector)();

/* Conversions Between Binary and Decimal Records */

void num2dec(f,x,d) /* d <-- x, according to format f */
    decform *f;
    extended x;
    decimal *d;
extended dec2num(d) /* returns d as extended */
    decimal *d;

/* Conversions Between Decimal Records and ASCII Strings */

void dec2str(f,d,s) /* s <-- d, according to format f */
    decform *f;
    decimal *d;
    char *s;
void str2dec(s,ix,d,vp) /* on input ix is starting index into s, */
    char *s; /* on output ix is one greater than index */
    short *ix,*vp; /* of last character of longest numeric */
    decimal *d; /* substring; boolean vp = "s beginning at */
    /* given ix is a valid numeric string or */
    /* a valid prefix of some numeric string" */

/* Arithmetic, Auxiliary, and Elementary Functions */

extended remainder(x,y,quo) /* IEEE remainder; quo <-- 7 low-order bits */
    extended x,y; /* of integer quotient x/y, */
    short *quo; /* -127 <= quo <= 127 */
extended rint(x) /* round to integral value */

```

```

    extended x;
extended scalb(n,x)      /* binary scale:  $x * 2^n$  */
    short n;
    extended x;
extended logb(x)         /* binary log: */
    extended x;         /* binary exponent of normalized x */
extended copysign(x,y)   /* y with sign of x */
    extended x,y;
extended nextfloat(x,y)  /* next float representation after */
    extended x,y;       /* (float) x in direction of (float) y */
extended nextdouble(x,y) /* next double representation after */
    extended x,y;       /* (double) x in direction of (double) y */
extended nextextended(x,y) /* next extended representation after x */
    extended x,y;       /* in direction of y */
extended log2(x)         /* base-2 log */
    extended x;
extended log1(x)         /*  $\log(1 + x)$  */
    extended x;
extended exp2(x)         /* base-2 exponential */
    extended x;
extended exp1(x)         /*  $\exp(x) - 1$  */
    extended x;
extended power(x,y)      /* general exponential:  $x^y$  */
    extended x,y;
extended ipower(x,i)     /* integer exponential:  $x^i$  */
    extended x;
    short i;
extended compound(r,n)    /* compound:  $(1 + r)^n$  */
    extended r,n;
extended annuity(r,n)    /* annuity:  $(1 - (1 + r)^{-n}) / r$  */
    extended r,n;
extended randomx(x)      /* returns next random number; updates x; */
    extended *x;         /* x integral,  $1 \leq x \leq 2^{31} - 2$  */

/* Inquiry Routines */

numclass classfloat(x) /* class of (float) x */
    extended x;
numclass classdouble(x) /* class of (double) x */
    extended x;
numclass classcomp(x) /* class of (comp) x */
    extended x;
numclass classextended(x) /* class of x */
    extended x;
long signum(x)          /* returns 0 for +, 1 for - */
    extended x;

/* Environment Access Routines */
/* An exception variable encodes the exceptions whose sum
/* is its value. */

void setexception(e,s) /* clr's e flags if s is 0, sets e flags */
    exception e;       /* otherwise; may cause halt */
    long s;

```



```

long testexception(e)      /* returns 1 if any e flag is set, */
    exception e;          /* returns 0 otherwise */
void sethalt(e,s)         /* disables e halts if s is 0, */
    exception e;          /* enables e halts otherwise */
    long s;
long testhalt(e)          /* returns 1 if any e halt is enabled, */
    exception e;          /* returns 0 otherwise */
void setround(r)           /* sets rounding direction to r */
    rounddir r;
    rounddir getround();  /* returns rounding direction */
void setprecision(p)       /* sets rounding precision to p */
    roundpre p;
    roundpre getprecision(); /* returns rounding precision */
void setenvironment(e)     /* sets environment to e */
    environment e;
void getenvironment(e)     /* e <-- environment */
    environment *e;
void procentry(e)          /* e <-- environment; */
    environment *e;        /* environment <-- IEEE default */
void procexit(e)           /* temp <-- exceptions; environment <-- e */
    environment e;         /* signals exceptions in temp */
    haltvector gethaltvector(); /* returns halt vector */
void sethaltvector(v)      /* halt vector <-- v */
    haltvector v;

/* Comparison Routine */

relop relation(x,y)        /* returns relation such that */
    extended x,y;         /* "x relation y" is true */

/* NaNs and Special Constants */

extended nan(c)            /* returns NaN with code c */
    unsigned char c;
extended inf();            /* infinity */
extended pi();             /* pi */

```

Description

The SANE functions fabs, sqrt, exp, log, tan, sin, cos, and atan are described in Chapter 3, "The Standard C Library."

These routines together with Apple's C language fully support the Standard Apple Numeric Environment (SANE). They provide a scrupulously conforming implementation of extended-precision IEEE Standard 754 floating-point arithmetic.

The Standard Apple Numeric Environment is documented in the *Apple Numerics Manual*.

Scrap—Scrap Manager

Synopsis

```
#include <Types.h>
#include <Scrap.h>

/* Type Definitions */

typedef struct ScrapStuff {
    long          scrapSize;
    Handle        scrapHandle;
    short         scrapCount;
    short         scrapState;
    StringPtr     scrapName;
} ScrapStuff, *PScrapStuff;

/* Getting Desk Scrap Information */

pascal PScrapStuff InfoScrap();

/* Keeping the Desk Scrap on the Disk */

pascal long UnloadScrap();
pascal long LoadScrap();

/* Writing to the Desk Scrap */

pascal long ZeroScrap();
pascal long PutScrap(length, theType, source)
    long length;
    ResType theType;
    Ptr source;

/* Reading From the Desk Scrap */

pascal long GetScrap(hDest, theType, offset)
    Handle hDest;
    ResType theType;
    long *offset;
```

Description

The Scrap Manager provides a mechanism for cutting and pasting between applications and desk accessories.

For more detailed information, see the Scrap Manager chapter of *Inside Macintosh*.

SCSI—SCSI Manager

Synopsis

```
#include <Types.h>
#include <SCSI.h>

/* Transfer Instruction Operation Codes */

#define scInc 1
#define scNoInc 2
#define scAdd 3
#define scMove 4
#define scLoop 5
#define scNop 6
#define scStop 7
#define scComp 8

typedef struct SCSIInstr {
    unsigned short scOpcode;
    unsigned long scParam1;
    unsigned long scParam2;
} SCSIInstr;

/* Routines */

pascal OSErr SCSIReset();
pascal OSErr SCSIReset();
pascal OSErr SCSIReset();
pascal OSErr SCSIReset();
pascal OSErr SCSIReset();
pascal OSErr SCSIReset();
pascal OSErr SCSIGet();
pascal OSErr SCSISelect(targetID)
pascal OSErr SCSIReset();
pascal OSErr SCSIReset();
    short targetID;
pascal OSErr SCSCmd(buffer, count)
    Ptr buffer;
    short count;
pascal OSErr SCSIRead(tibPtr)
    Ptr tibPtr;
pascal OSErr SCIRBlind(tibPtr)
    Ptr tibPtr;
pascal OSErr SCSIWrite(tibPtr)
    Ptr tibPtr;
pascal OSErr SCSIWBlind(tibPtr)
    Ptr tibPtr;
pascal OSErr SCSIComplete(stat, message, wait)
    short *stat,*message;
    unsigned long wait;
pascal short SCSIStat();
```

Description The SCSI Manager controls the exchange of information between a Macintosh and peripheral devices connected through the Small Computer Standard Interface (SCSI). For more detailed information, see the SCSI Manager chapter of *Inside Macintosh*, Volume 4.

SegLoad—Segment Loader

Synopsis

```
#include <Types.h>
#include <SegLoad.h>

/* Message Returned by CountAppFiles */

#define appOpen 0 /* open the document(s) */
#define appPrint 1 /* print the document(s) */

typedef struct AppFile {
    short vRefNum; /* volume reference number */
    OSType fType; /* file type */
    short versNum; /* version number */
    Str255 fName; /* file name */
} AppFile;

/* Routines */

void CountAppFiles(message, count)
    short *message;
    short *count;
void GetAppFiles(index, theFile)
    short index;
    AppFile *theFile;
void ClrAppFiles(index)
    short index;
void GetAppParms(apName, apRefNum, apParam)
    char *apName;
    short *apRefNum;
    Handle *apParam;
pascal void UnloadSeg(routineAddr)
    Ptr routineAddr;
pascal void ExitToShell();
```

Description

The Segment Loader is the part of the Macintosh Operating System that lets you divide your application into several parts and have only some of them in memory at a time. When an application starts up, the Segment Loader also provides it with a list of files to open or print.

For more detailed information, see the Segment Loader chapter of *Inside Macintosh*.

Serial—Serial Drivers

Synopsis

```
#include <Types.h>
#include <Serial.h>

/* Driver Reset Information */

#define baud300      380 /* 300 baud */
#define baud600      189 /* 600 baud */
#define baud1200     94  /* 1200 baud */
#define baud1800     62  /* 1800 baud */
#define baud2400     46  /* 2400 baud */
#define baud3600     30  /* 3600 baud */
#define baud4800     22  /* 4800 baud */
#define baud7200     14  /* 7200 baud */
#define baud9600     10  /* 9600 baud */
#define baud19200    4   /* 19200 baud */
#define baud57600    0   /* 57600 baud */
#define stop10       16384 /* 1 stop bit */
#define stop15       (-32768) /* 1.5 stop bits */
#define stop20       (-16384) /* 2 stop bits */
#define noParity      0 /* no parity */
#define oddParity     4096 /* odd parity */
#define evenParity    12288 /* even parity */
#define data5         0 /* 5 data bits */
#define data6         2048 /* 6 data bits */
#define data7         1024 /* 7 data bits */
#define data8         3072 /* 8 data bits */

/* Masks for Changes That Cause Events to Be Posted */

#define ctsEvent      32 /* set if CTS change will cause event */
                        /* to be posted */

#define breakEvent    128 /* set if break status change will */
                        /* cause event to be posted */

/* Indication That XOFF Char Was Sent */

#define xOffWasSent    0x80

/* Indication That DTR Is Negated */

#define dtrNegated     0x40

typedef enum {
    sPortA, /* modem port */
    sPortB, /* printer port */
} SPortSel;
```

```

typedef struct SerShk {
    char          fXOn;          /* XOn/XOff output flow control flag */
    char          fCTS;          /* CTS hardware handshake flag */
    unsigned char xOn;           /* XOn character */
    unsigned char xOff;          /* XOff character */
    char          errs;          /* errors that cause abort */
    char          evts;          /* status changes that cause events */
    char          fInX;          /* XOn/XOff input flow control flag */
    char          fDTR;          /* DTR input flow control flag */
} SerShk;

typedef struct SerStaRec {
    char          cumErrs;        /* cumulative errors */
    char          xOffSent;       /* XOff sent as input flow control */
    char          rdPend;         /* read pending flag */
    char          wrPend;         /* write pending flag */
    char          ctsHold;        /* CTS flow control hold flag */
    char          xOffHold;       /* XOff received as output flow
                                control */
} SerStaRec;

/* Opening and Closing the RAM Serial Driver */

OSErr RAMSDOpen(whichPort)
    SPortSel whichPort;
void RAMSDClose(whichPort)
    SPortSel whichPort;

/* Changing Serial Driver Information */

OSErr SerReset(refNum,serConfig)
    short refNum;
    short serConfig;
OSErr SerSetBuf(refNum,serBPtr,serBLen)
    short refNum;
    Ptr serBPtr;
    short serBLen;
OSErr SerHShake(refNum,flags);
    short refNum;
    SerShk *flags;
OSErr SerSetBrk(refNum)
    short refNum;
OSErr SerClrBrk(refNum)
    short refNum;

/* Getting Serial Driver Information */

OSErr SerGetBuf(refNum,count)
    short refNum;
    long *count;
OSErr SerStatus(refNum,serSta)
    short refNum;
    SerStaRec *serSta;

```

Description

The RAM Serial Driver and the ROM Serial Driver are Macintosh device drivers for handling asynchronous serial communication between a Macintosh application and serial devices.

For more detailed information, see the Serial Drivers chapter of *Inside Macintosh*.

Sound—Sound Driver

Synopsis

```
#include <Types.h>
#include <Sound.h>

/* Mode Values for Synthesizers */

#define swMode      (-1) /* square-wave synthesizer */
#define ftMode       1 /* four-tone synthesizer */
#define ffMode       0 /* free-form synthesizer */

/* Free-Form Synthesizer */

typedef unsigned char FreeWave[30001];
typedef struct FFSynthRec {
    short          mode;          /* always ffMode */
    Fixed          count;         /* "sizing" factor */
    FreeWave       waveBytes;     /* waveform description */
} FFSynthRec, *FFSynthPtr;

/* Square-Wave Synthesizer */

typedef struct Tone {
    short          count;         /* frequency */
    short          amplitude;     /* amplitude, 0-255 */
    short          duration;      /* duration in ticks */
} Tone;

typedef Tone Tones[5001];
typedef struct SWSynthRec {
    short          mode;          /* always swMode */
    Tones          triplets;      /* sounds */
} SWSynthRec, *SWSynthPtr;

/* Four-Tone Synthesizer */

typedef unsigned char Wave[256];
typedef Wave *WavePtr;
typedef struct FTSoundRec {
    short          duration;      /* duration in ticks */
    Fixed          sound1Rate;    /* tone 1 cycle rate */
    long           sound1Phase;   /* tone 1 byte offset */
    Fixed          sound2Rate;    /* tone 2 cycle rate */
    long           sound2Phase;   /* tone 2 byte offset */
    Fixed          sound3Rate;    /* tone 3 cycle rate */
    long           sound3Phase;   /* tone 3 byte offset */
    Fixed          sound4Rate;    /* tone 4 cycle rate */
    long           sound4Phase;   /* tone 4 byte offset */
}
```

```

    WavePtr      sound1Wave;    /* tone 1 wave form */
    WavePtr      sound2Wave;    /* tone 2 wave form */
    WavePtr      sound3Wave;    /* tone 3 wave form */
    WavePtr      sound4Wave;    /* tone 4 wave form */
} FTSoundRec, *FTSndRecPtr;

typedef struct FTSynthRec {
    short          mode;        /* always ftMode */
    FTSndRecPtr    sndRec;      /* tones to play */
} FTSynthRec, *FTSynthPtr;

/* Routines [Not in ROM] */

void StartSound(synthRec,numBytes,completionRtn)
    Ptr synthRec;
    long numBytes;
    ProcPtr completionRtn;
void StopSound();
Boolean SoundDone();
void GetSoundVol(level)
    short *level;
void SetSoundVol(level)
    short level

```

Description

The Sound Driver is a Macintosh device driver for handling sound and music generation in a Macintosh application.

For more detailed information, see the Sound Driver chapter of *Inside Macintosh*.

Strings—string conversions

Synopsis

```
#include <Strings.h>
```

```
/* Routines */
```

```
char *c2pstr(s)
```

```
    char *s;
```

```
char *p2cstr(s)
```

```
    char *s;
```

Description

Function `c2pstr` converts `s` from a C string to a Pascal string. Function `p2cstr` converts `s` from a Pascal string to a C string. Both conversions are done in place. For convenience, `c2pstr` and `p2cstr` return `s` as their function result. Both functions will accept `nil` as their parameter and do nothing.

Pascal strings begin with a length byte. C strings are terminated by a zero byte. The macro `String` is defined in file `Types.h`.

TextEdit—text-editing routines

Synopsis

```
#include <Types.h>
#include <TextEdit.h>

/* Text justification */

#define teJustLeft      0
#define teJustCenter    1
#define teJustRight    (-1)

typedef struct Terec {
    Rect      destRect;      /* destination rectangle */
    Rect      viewRect;      /* view rectangle */
    Rect      selRect;       /* select rectangle */
    short     lineHeight;    /* current font line-height */
    short     fontAscent;    /* current font ascent */
    Point     selPoint;       /* selection point (mouseLoc) */
    short     selStart;       /* selection start */
    short     selEnd;         /* selection end */
    short     active;         /* != 0 if active */
    ProcPtr   wordBreak;     /* word-break routine */
    ProcPtr   clickLoop;     /* click-loop routine */
    long      clickTime;     /* time of first click */
    short     clickLoc;      /* char. location of click */
    long      caretTime;     /* time for next caret blink */
    short     caretState;    /* on/active booleans */
    short     just;          /* fill style */
    short     teLength;      /* length of text below */
    Handle     hText;        /* handle to actual text */
    short     recalBack;     /* != 0 if recal in background */
    short     recalLines;    /* line being recalculated */
    short     klikStuff;     /* click stuff (internal) */
    short     crOnly;        /* set to -1 if CR Line breaks only */
    short     txFont;        /* text Font */
    Style     txFace;        /* text Face */
    short     txMode;        /* text Mode */
    short     txSize;        /* text Size */
    struct GrafPort *inPort; /* GrafPort */
    ProcPtr   highHook;      /* highlighting hook */
    ProcPtr   caretHook;     /* caret hook */
    short     nLines;        /* number of lines */
    short     lineStarts[16001]; /* line starts */
} Terec, *TEPtr, **TEHandle;

typedef char Chars[32001];

typedef Chars *CharsPtr, **CharsHandle;
```

```

/* Initialization and Allocation */

pascal void TEInit();
pascal TEHandle TENew(destRect,viewRect)
    Rect *destRect, *viewRect;
pascal void TEDispose(hTE)
    TEHandle hTE;

/* Accessing the Text of an Edit Record */

pascal void TEText(text,length,hTE)
    Ptr text;
    long length;
    TEHandle hTE;
pascal CharsHandle TEGetText(hTE)
    TEHandle hTE;

/* Insertion Point and Selection Range */

pascal void TEIdle(hTE)
    TEHandle hTE;
void TEClick(pt,extend,hTE)
    Point *pt;
    Boolean extend;
    TEHandle hTE;
pascal void TETextSelect(selStart,selEnd,hTE)
    long selStart,selEnd;
    TEHandle hTE;
pascal void TEActivate(hTE)
    TEHandle hTE;
pascal void TEDeactivate(hTE)
    TEHandle hTE;

/* Editing */

pascal void TEKey(key,hTE)
    short key;
    TEHandle hTE;
pascal void TECut(hTE)
    TEHandle hTE;
pascal void TECopy(hTE)
    TEHandle hTE;
pascal void TEPaste(hTE)
    TEHandle hTE;
pascal void TEDelete(hTE)
    TEHandle hTE;
pascal void TEInsert(text,length,hTE)
    Ptr text;
    long length;
    TEHandle hTE;

```

```

/* Text Display and Scrolling */

pascal void TETSetJust(just,hTE)
    short just;
    TEHandle hTE;
pascal void TEUpdate(rUpdate,hTE)
    Rect *rUpdate;
    TEHandle hTE;
pascal void TextBox(text,length,box,just)
    Ptr text;
    long length;
    Rect *box;
    short just;
pascal void TEScroll(dh,dv,hTE)
    short dh,dv;
    TEHandle hTE;
pascal void TEselView(hTE)
    TEHandle hTE;
pascal void TEPinScroll(dh,dv,hTE)
    short dh;
    short dv;
    TEHandle hTE;
pascal void TEAutoView(pAuto,hTE)
    Boolean pAuto;
    TEHandle hTE;

/* Scrap Handling [Not in ROM] */

OSErr TETFromScrap();
OSErr TETToScrap();
Handle TEScrapHandle();
long TETGetScrapLen();
void TETSetScrapLen(length)
    long length;

/* Advanced Routines */

void SetWordBreak(wBrkProc,hTE)
    ProcPtr wBrkProc;
    TEHandle hTE;
void SetClikLoop(clikProc,hTE)
    ProcPtr clikProc;
    TEHandle hTE;
pascal void TECalText(hTE)
    TEHandle hTE;

```

User routines /* Word Break Routine */

```

pascal Boolean MyWordBreak(text,charPos)
    Ptr text;
    short charPos;

```

```
/* Click Loop Routine */
```

```
pascal Boolean MyClickLoop();
```

Description

The TextEdit package provides basic text formatting and editing.
For more detailed information, see the TextEdit chapter of *Inside Macintosh*.

Note

The user routines `highHook` and `caretHook` are called with register conventions and therefore can't be C routines.

Time—Time Manager

Synopsis

```
#include <Types.h>
#include <OSUtils.h>
#include <Time.h>

typedef struct TMTask {
    struct QElem *qLink; /* next queue entry */
    short qType; /* queue type */
    ProcPtr tmAddr; /* pointer to routine */
    short tmCount; /* reserved */
} TMTask;

pascal void InsTime(tmTaskPtr)
    TMTask *tmTaskPtr;
pascal void PrimeTime(tmTaskPtr, count)
    TMTask *tmTaskPtr;
    long count;
pascal void RmvTime(tmTaskPtr)
    TMTask *tmTaskPtr;
```

Description

The Time Manager is the part of the operating system that lets you schedule a routine to be executed after a given number of milliseconds have elapsed.

For more detailed information, see the Time Manager chapter of *Inside Macintosh*, Volume 4.

ToolUtils—Toolbox Utilities

Synopsis

```
#include <Types.h>
#include <QuickDraw.h>
#include <ToolUtils.h>

/* Resource ID of Standard Pattern List */

#define sysPatListID 0
#define iBeamCursor 1 /* text selection */
#define crossCursor 2 /* drawing graphics */
#define plusCursor 3 /* cell selection */
#define watchCursor 4 /* indicating long delay */

typedef struct Int64Bit {
    long hiLong;
    long loLong;
} Int64Bit;

typedef struct Cursor *CursPtr, **CursHandle;

typedef Pattern *PatPtr, **PatHandle;

/* Fixed-Point Arithmetic */

pascal Fixed FixRatio( numer, denom)
    short numer, denom;
pascal Fixed FixMul( a, b)
    Fixed a, b;
pascal short FixRound( x)
    Fixed x;

/* String Manipulation */

StringHandle NewString( theString)
    char *theString;
void SetString( h, theString)
    StringHandle h;
    char *theString;
pascal StringHandle GetString( stringID)
    short stringID;
void GetIndString( theString, strListID, index)
    char *theString;
    short strListID;
    short index;
```

```

/* Byte Manipulation */

pascal long Munger(h,offset,ptr1,len1,ptr2,len2)
    Handle h;
    long offset;
    Ptr ptr1;
    long len1;
    Ptr ptr2;
    long len2;
pascal void PackBits(srcPtr,dstPtr,srcBytes)
    Ptr *srcPtr,*dstPtr;
    short srcBytes;
pascal void UnpackBits(srcPtr,dstPtr,dstBytes)
    Ptr *srcPtr,*dstPtr;
    short dstBytes;

/* Bit Manipulation */

pascal Boolean BitTst(bytePtr,bitNum)
    Ptr bytePtr;
    long bitNum;
pascal void BitSet(bytePtr,bitNum)
    Ptr bytePtr;
    long bitNum;
pascal void BitClr(bytePtr,bitNum)
    Ptr bytePtr;
    long bitNum;

/* Logical Operations */

pascal long BitAnd(value1,value2)
    long value1,value2;
pascal long BitOr(value1,value2)
    long value1,value2;
pascal long BitXor(value1,value2)
    long value1,value2;
pascal long BitNot(value)
    long value;
pascal long BitShift(value,count)
    long value;
    short count;

/* Operations on Long Integers */

pascal short HiWord(x)
    long x;
pascal short LoWord(x)
    long x;
pascal void LongMul(a,b,dest)
    long a,b;
    Int64Bit *dest;

```

```

/* Graphics Utilities */

void ScreenRes (scrnHRes, scrnVRes)
    short *scrnHRes, *scrnVRes;
pascal Handle GetIcon (iconID)
    short iconID;
pascal void PlotIcon (theRect, theIcon)
    Rect *theRect;
    Handle theIcon;
pascal PatHandle GetPattern (patID)
    short patID;
void GetIndPattern (thePattern, patListID, index)
    Pattern thePattern;
    short patListID;
    short index;
pascal CursHandle GetCursor (cursorID)
    short cursorID;
void ShieldCursor (shieldRect, offsetPt)
    Rect *shieldRect;
    Point *offsetPt;
pascal struct Picture **GetPicture (picID)
    short picID;

/* Miscellaneous Utilities */

long DeltaPoint (ptA, ptB)
    Point *ptA, *ptB;
pascal Fixed SlopeFromAngle (angle)
    short angle;
pascal short AngleFromSlope (slope)
    Fixed slope;

```

- Description** The Toolbox Utilities provide fixed-point arithmetic; string, byte, and bit manipulation; logical operations; and some graphics utilities.
- For more detailed information, see the *ToolBox Utilities* chapter of *Inside Macintosh*.
- Note** The *FixMath* section in this chapter describes additional fixed-point arithmetic routines.
- Warning** *NewString* and *GetString* return handles to Pascal strings. *NewString*, *SetString*, and *GetIndString* take a C string as their parameter and convert it to a Pascal string before storing it in memory.

Types—common defines and types

Synopsis

```
#include <Types.h>
#define nil 0
#define NULL 0
#define noErr 0

typedef enum {false,true} Boolean;
typedef char *Ptr;
typedef Ptr *Handle;
typedef long (*ProcPtr)();
typedef ProcPtr *ProcHandle;
typedef long Fixed;
typedef long Fract;
typedef unsigned long ResType;
typedef long OSType;
typedef short OSErr;
typedef short Style;
typedef struct Point {
    short v;
    short h;
} Point;
typedef struct Rect {
    short top;
    short left;
    short bottom;
    short right;
} Rect;

/* Pascal String Macro */

#define String(size) struct {\
    unsigned char length; unsigned char text[size];}

/* Inside Macintosh String Definitions */

typedef String(255) Str255, *StringPtr, **StringHandle;
```

Description

These defines and types are shared by several Macintosh libraries.

The define `String` approximates Pascal strings. It creates a struct, not an array. Remember to use `&` when passing structs as parameters.

Windows—Window Manager

Synopsis

```
#include <Types.h>
#include <QuickDraw.h>
#include <Windows.h>

/* Window Definition IDs */

#define documentProc 0
#define dBoxProc 1
#define plainDBox 2
#define altDBoxProc 3
#define noGrowDocProc 4
#define zoomDocProc 8
#define rDocProc 16

/* Window Class, in windowKind Field of Window Record */

#define dialogKind 2
#define userKind 8

/* Values Returned by FindWindow */

#define inDesk 0
#define inMenuBar 1
#define inSysWindow 2
#define inContent 3
#define inDrag 4
#define inGrow 5
#define inGoAway 6
#define inZoomIn 7
#define inZoomOut 8

/* Axis Constraints for DragGrayRgn */

#define noConstraint 0
#define hAxisOnly 1
#define vAxisOnly 2

/* Messages to Window Definition Function */

#define wDraw 0
#define wHit 1
#define wCalcRgns 2
#define wNew 3
#define wDispose 4
#define wGrow 5
#define wDrawGIcon 6
```

```

/* Values Returned by Window Definition Function's Hit Routine */

#define    wNoHit        0
#define    wInContent    1
#define    wInDrag       2
#define    wInGrow       3
#define    wInGoAway     4
#define    wInZoomIn     5
#define    wInZoomOut    6

/* Resource ID of Desktop Pattern */

#define    deskPatID     16

typedef GrafPtr WindowPtr;
typedef struct WindowRecord {
    GrafPort        port;
    short           windowKind;
    Boolean          visible;
    Boolean          hilited;
    Boolean          goAwayFlag;
    Boolean          spareFlag;
    RgnHandle        strucRgn;
    RgnHandle        contRgn;
    RgnHandle        updateRgn;
    Handle           windowDefProc;
    Handle           dataHandle;
    StringHandle     titleHandle;
    short           titleWidth;
    struct ControlRecord **controlList;
    struct WindowRecord *nextWindow;
    PicHandle        windowPic;
    long            refCon;
} WindowRecord, *WindowPeek;

typedef struct WStateData {
    Rect             userState;
    Rect             stdState;
} WStateData;

/* Initialization and Allocation */

pascal void InitWindows();
pascal void GetWMgrPort(wPort)
    GrafPtr *wPort;
WindowPtr NewWindow(wStorage, boundsRect, title, visible, procID, behind,
    goAwayFlag, refCon)
    Ptr wStorage;
    Rect *boundsRect;
    char *title;
    Boolean visible;
    short procID;
    WindowPtr behind;

```

```

    Boolean goAwayFlag;
    long refCon;
    pascal WindowPtr GetNewWindow(windowID,wStorage,behind)
        short windowID;
        Ptr wStorage;
        WindowPtr behind;
    pascal void CloseWindow(theWindow)
        WindowPtr theWindow;
    pascal void DisposeWindow(theWindow)
        WindowPtr theWindow;

/* Window Display */

void SetWTitle(theWindow,title)
    WindowPtr theWindow;
    char *title;
void GetWTitle(theWindow,title)
    WindowPtr theWindow;
    char *title;
pascal void SelectWindow(theWindow)
    WindowPtr theWindow;
pascal void HideWindow(theWindow)
    WindowPtr theWindow;
pascal void ShowWindow(theWindow)
    WindowPtr theWindow;
pascal void ShowHide(theWindow,showFlag)
    WindowPtr theWindow;
    Boolean showFlag;
pascal void HiliteWindow(theWindow,fHilite)
    WindowPtr theWindow;
    Boolean fHilite;
pascal void BringToFront(theWindow)
    WindowPtr theWindow;
pascal void SendBehind(theWindow,behindWindow)
    WindowPtr theWindow,behindWindow;
pascal WindowPtr FrontWindow();
pascal void DrawGrowIcon(theWindow)
    WindowPtr theWindow;

/* Mouse Location */

short FindWindow(thePt,whichWindow)
    Point *thePt;
    WindowPtr *whichWindow;
Boolean TrackGoAway(theWindow,thePt)
    WindowPtr theWindow;
    Point *thePt;
Boolean TrackBox(theWindow,thePt,partCode)
    WindowPtr theWindow;
    Point *thePt;
    short partCode;

```

```

/* Window Movement and Sizing */

pascal void MoveWindow(theWindow,hGlobal,vGlobal,front)
    WindowPtr theWindow;
    short hGlobal,vGlobal;
    Boolean front;
void DragWindow(theWindow,startPt,boundsRect)
    WindowPtr theWindow;
    Point *startPt;
    Rect *boundsRect;
long GrowWindow(theWindow,startPt,sizeRect)
    WindowPtr theWindow;
    Point *startPt;
    Rect *sizeRect;
pascal void SizeWindow(theWindow,w,h,fUpdate)
    WindowPtr theWindow;
    short w,h;
    Boolean fUpdate;
pascal void ZoomWindow(theWindow,partCode,front)
    WindowPtr theWindow;
    short partCode;
    Boolean front;

/* Update Region Maintenance */

pascal void InvalRect(badRect)
    Rect *badRect;
pascal void InvalRgn(badRgn)
    RgnHandle badRgn;
pascal void ValidRect(goodRect)
    Rect *goodRect;
pascal void ValidRgn(goodRgn)
    RgnHandle goodRgn;
pascal void BeginUpdate(theWindow)
    WindowPtr theWindow;
pascal void EndUpdate(theWindow)
    WindowPtr theWindow;

/* Miscellaneous Routines */

pascal void SetWRefCon(theWindow,data)
    WindowPtr theWindow;
    long data;
pascal long GetWRefCon(theWindow)
    WindowPtr theWindow;
pascal void SetWindowPic(theWindow,pic)
    WindowPtr theWindow;
    PicHandle pic;
pascal PicHandle GetWindowPic(theWindow)
    WindowPtr theWindow;
long PinRect(theRect,thePt)
    Rect *theRect;
    Point *thePt;

```



```

long DragGrayRgn(theRgn, startPt, limitRect, slopRect, axis, actionProc)
    RgnHandle theRgn;
    Point *startPt;
    Rect *limitRect, *slopRect;
    short axis;
    ProcPtr actionProc;

```

```

/* Low-Level Routines */

```

```

pascal Boolean CheckUpdate(theEvent)
    struct EventRecord *theEvent;
pascal void ClipAbove(window)
    WindowPeek window;
pascal void SaveOld(window)
    WindowPeek window;
pascal void DrawNew(window, update)
    WindowPeek window;
    Boolean update;
pascal void PaintOne(window, clobberedRgn)
    WindowPeek window;
    RgnHandle clobberedRgn;
pascal void PaintBehind(startWindow, clobberedRgn)
    WindowPeek startWindow;
    RgnHandle clobberedRgn;
pascal void CalcVis(window)
    WindowPeek window;
pascal void CalcVisBehind(startWindow, clobberedRgn)
    WindowPeek startWindow;
    RgnHandle clobberedRgn;

```

User routines

```

pascal MyAction();
pascal long MyWindow(varCode, theWindow, message, param)
    short varCode;
    WindowPtr theWindow;
    short message;
    long param;

```

Description

The Window Manager provides routines for creating and manipulating windows. For more detailed information, see the Window Manager chapter of *Inside Macintosh*.



Appendix A



Calling Conventions

MPW C uses two different function-calling conventions: C calling conventions and Pascal-compatible calling conventions.

C calling conventions

This section describes the normal C calling conventions. It explains how function parameters are passed, how function results are returned, and how registers are saved across function calls. This information is useful when writing calls between C and assembly language.

Parameters

Parameters to C functions are evaluated from right to left and are pushed onto the stack in the order they are evaluated. Characters, integers, and enumerated types are passed as sign-extended 32-bit values. Pointers and arrays are passed as 32-bit addresses. Types float, double, comp, and extended are passed as extended 80-bit values. Structures are also passed on the stack. Their size is rounded up to a multiple of 16 bits (2 bytes). If rounding occurs, the unused storage has the highest memory address. The caller removes the parameters from the stack.

Function results

Characters, integers, enumerated types, and pointers are returned as sign-extended 32-bit values in register D0. Types float, double, comp, and extended are returned as extended values in registers D0, D1, and A0. The low-order 16 bits of D0 contain the sign and exponent bits, register D1 contains the high-order 32 bits of the significand, and register A0 contains the low-order 32 bits of the significand. Structure values are returned as a 32-bit pointer in register D0. The pointer contains the address of a static variable into which the result is copied before returning. This implementation of structure function results is not reentrant.

Register conventions

Registers D0, D1, A0, and A1 are scratch registers that are not preserved by C functions. All other registers are preserved. Register A5 is the global frame pointer, register A6 is the local frame pointer, and register A7 is the stack pointer. Local stack frames are not necessarily created for simple functions.

Pascal-compatible calling conventions

This section describes the MPW C conventions for calling Pascal functions and for calling C functions that use Pascal-compatible calling conventions. These conventions differ from the normal C calling conventions described earlier in this appendix; they also differ from the calling conventions used by the Pascal Compiler. This section explains how function parameters are passed, how function results are returned, and how registers are saved across function calls.

Parameters

Parameters to Pascal-compatible functions are evaluated left to right and are pushed onto the stack in the order they are evaluated. Characters and enumerated types whose literal values fall in the range of types `char` or `unsigned char` are pushed as bytes. (This requires a 16-bit word on the stack. The value is in the high-order 8 bits; the low-order 8 bits are unused.) Short values and enumerated types whose literal values fall in the range of types `short` or `unsigned short` are passed as 16-bit values. `int` and `long` values and the remaining enumerated types are passed as 32-bit values. Pointers and arrays are passed as 32-bit addresses. SANE types `float`, `double`, `comp`, and `extended` are passed as extended 80-bit values; this doesn't correspond to the Pascal Compiler's calling conventions, however, so a compiler warning is given. (Table 2-2 shows the recommended way to pass SANE-type values to Pascal.) Structures are also passed by value on the stack, and also cause a compiler warning. Their size is rounded up to a multiple of 16 bits (2 bytes). If rounding occurs, the unused storage has the highest memory address. The function being called removes the parameters from the stack.

Function results

Function results are returned on the stack. Stack space for the function result is reserved by the caller before pushing any parameters. Characters and enumerated types whose literal values fall in the range of types `char` or `unsigned char` are returned as bytes. (This requires a 16-bit word on the stack. The value is in the high-order 8 bits; the low-order 8 bits are unused.) Short values and enumerated types whose literal values fall in the range of types `short` or `unsigned short` are returned as 16-bit values. `int` and `long` values and the remaining enumerated types are returned as 32-bit values. Pointers are returned as 32-bit addresses. Arrays may not be returned as function results. Results of type `float` are returned as 32-bit values. For structures and types `double`, `comp`, and `extended`, the caller pushes the address of a structure, `double`, `comp`, or `extended` (respectively) in the function-result location on the stack. The procedure being called stores the result at this address. The caller removes the function results from the stack.

For structure results, if the Pascal function returns a structure of greater than 4 bytes, the caller pushes a pointer to a result space before pushing any results. If the structure is 4 bytes or less, the caller reserves 2 or 4 bytes on the stack for it.

Register conventions

Registers D0, D1, D2, A0, and A1 are scratch registers. Scratch registers are not preserved by Pascal-compatible functions. All other registers are preserved. Register A5 is the global frame pointer, register A6 is the local frame pointer, and register A7 is the stack pointer.

Appendix B

Files Supplied With MPW C

MPW C is intended for use with the Macintosh Programmer's Workshop. The files listed below are on the two MPW C release disks: the first disk contains the C Compiler, and the second disk contains sample programs, the Standard C Library, and the Macintosh Interface Libraries. These files may be used directly from the release disks or copied onto a hard disk.

Disk C1: contains a single file, the Green Hills Software C Compiler for Macintosh.

Disk C2: contains files in three directories. Directory CExamples: contains instructions, a make file, and source for the example programs. CIncludes: contains header files for use with the Standard C Library and Macintosh Interface Libraries. CLibraries: contains library object files.

C Compiler (disk C1)

| Filename | Comments |
|----------|----------------|
| C | MPW C Compiler |

C Compiler files (disk C2:CExamples)

| Filename | Comments |
|----------------|--|
| Instructions.c | instructions for building examples |
| MakeFile.c | make file for building examples |
| Sample.c | source for Sample application |
| Sample.r | resource specifications for Sample application |

| | |
|----------|---|
| Count.c | source for Count tool |
| Stubs.c | source for dummy library routines |
| Memory.c | source for Memory desk accessory |
| Memory.r | resource specifications for Memory desk accessory |

Libraries (disk C2:CIncludes)

| Filename | Comments |
|-------------|---|
| AppleTalk.h | AppleTalk header file |
| Controls.h | Control Manager header file |
| CType.h | character types header file |
| Desk.h | Desk Manager header file |
| Devices.h | Device Manager header file |
| Dialogs.h | Dialog Manager header file |
| DiskInit.h | Disk Initialization header file |
| Disks.h | Disk Driver header file |
| ErrNo.h | Standard C Library error numbers |
| Errors.h | Macintosh Interface Libraries error numbers |
| Events.h | Toolbox Event Manager header file |
| FCntl.h | file controls header file |
| Files.h | File Manager header file |
| FixMath.h | Fixed Point Math header file |
| Fonts.h | Font Manager header file |
| Graf3D.h | Graf3D header file |
| IOCtl.h | I/O Control header file |
| Lists.h | List Manager header file |
| Math.h | mathematical functions header file |
| Memory.h | Memory Manager header file |
| Menus.h | Menu Manager header file |
| OSEvents.h | Operating System Event Manager header file |
| OSUtils.h | Operating System Utilities header file |
| Packages.h | Packages header file |
| Printing.h | Print Manager header file |
| QuickDraw.h | QuickDraw header file |
| Resources.h | Resource Manager header file |
| Retrace.h | Vertical Retrace header file |
| SANE.h | SANE header file |
| Scrap.h | Scrap Manager header file |
| SCSI.h | SCSI Manager header file |
| SegLoad.h | Segment Loader header file |
| Serial.h | Serial Driver header file |
| SetJump.h | set jmp header file |
| Signal.h | signal handler header file |
| Sound.h | Sound Driver header file |
| StdIO.h | Standard I/O header file |
| Strings.h | string conversion header file |

| | |
|-------------|------------------------------------|
| TextEdit.h | TextEdit header file |
| Time.h | Time Manager header file |
| ToolUtils.h | Toolbox Utilities header file |
| Types.h | common types header file |
| Values.h | arithmetic values header file |
| VarArgs.h | variable argument list header file |
| Windows.h | Window Manager header file |

Object files (disk C2:CLibraries)

| Filename | Comments |
|--------------|---------------------------------|
| CInterface.o | C Macintosh Interface Libraries |
| CRuntime.o | C runtime library |
| CSANELib.o | C SANE library |
| Math.o | mathematical functions library |
| StdCLib.o | Standard C Library |



Appendix C



The Library Index

The Library Index contains an index entry for every define, type, enumeration literal, global variable, macro, and function defined in the Standard C Library and the Macintosh Interface Libraries. The manual pages are organized alphabetically within their chapter with one exception: the manual page "Error Numbers" in the Standard C Library chapter appears first.

- Column 1 of the Library Index contains an alphabetical list of the index entries.
- Column 2 specifies the type of declaration—for example, "define"—for the index entry.
- Column 3 contains the name of the manual page on which documentation for the index entry can be found.
- "(C)" following the manual page—for example, "printf(C)"—means look in Chapter 3, "The Standard C Library."
- Nothing following the manual page—for example, "AppleTalk"—means look in Chapter 4, "The Macintosh Interface Libraries."

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|----------------|----------|-------------|----------------|----------|-------------|
| acorevDate | literal | Packages | AppFile | type | SegLoad |
| ABCallType | type | AppleTalk | appleMark | define | Fonts |
| acortErr | define | Errors | applFont | define | Fonts |
| ABProtoType | type | AppleTalk | ApplicZone | function | Memory |
| abs | function | abs(C) | appOpen | define | SegLoad |
| acos | function | trig(C) | appPrint | define | SegLoad |
| activateEvt | define | Events | asin | function | trig(C) |
| activeFlag | define | Events | atan | function | trig(C) |
| activMask | define | Events | atan2 | function | trig(C) |
| AddDrive | function | Files | ATATPRec | type | AppleTalk |
| AddPt | function | Quickdraw | ATATPRecHandle | type | AppleTalk |
| AddrBlock | type | AppleTalk | ATATPRecPtr | type | AppleTalk |
| addRefFailed | define | Errors | ATDDPRec | type | AppleTalk |
| addResFailed | define | Errors | ATDDPRecHandle | type | AppleTalk |
| AddResMenu | function | Menus | ATDDPRecPtr | type | AppleTalk |
| AddResource | function | Resources | athens | define | Fonts |
| Alert | function | Dialogs | ATLAPRec | type | AppleTalk |
| AlertTemplate | type | Dialogs | ATLAPRecHandle | type | AppleTalk |
| AlertTHndl | type | Dialogs | ATLAPRecPtr | type | AppleTalk |
| AlertTPtr | type | Dialogs | ATNBPreC | type | AppleTalk |
| Allocate | function | Files | ATNBPreCHandle | type | AppleTalk |
| alphaLock | define | Events | ATNBPreCPtr | type | AppleTalk |
| altDBoxProc | define | Windows | atof | function | atof(C) |
| AngleFromSlope | function | ToolUtils | atoi | function | atoi(C) |
| annuity | function | SANE | atol | function | atoi(C) |
| applEvt | define | Events | ATPAddrRsp | function | AppleTalk |
| applMask | define | Events | atpBadRsp | define | AppleTalk |
| app2Evt | define | Events | ATPCloseSocket | function | AppleTalk |
| app2Mask | define | Events | ATPGetRequest | function | AppleTalk |
| app3Evt | define | Events | atpLenErr | define | Errors |
| app3Mask | define | Events | ATPLoad | function | AppleTalk |
| app4Evt | define | Events | ATPOpenSocket | function | AppleTalk |
| app4Mask | define | Events | atpProto | literal | AppleTalk |
| AppendMenu | function | Menus | ATPReqCancel | function | AppleTalk |

| Identifier | Type | Manual page |
|---------------|----------|-------------|
| ATPRequest | function | AppleTalk |
| ATPResponse | function | AppleTalk |
| ATPRspCancel | function | AppleTalk |
| atpSize | define | AppleTalk |
| ATPSndRequest | function | AppleTalk |
| ATPSndRsp | function | AppleTalk |
| ATPUnLoad | function | AppleTalk |
| autoKey | define | Events |
| autoKeyMask | define | Events |
| autoTrack | define | Controls |
| BackColor | function | Quickdraw |
| BackPat | function | Quickdraw |
| bufATPSkt | define | AppleTalk |
| badBtSlpErr | define | Errors |
| badBuffNum | define | AppleTalk |
| badCksmErr | define | Errors |
| badDBtSlp | define | Errors |
| badDCKsum | define | Errors |
| badMDBErr | define | Errors |
| badMovErr | define | Errors |
| badUnitErr | define | Errors |
| baud1200 | define | Serial |
| baud1300 | define | Serial |
| baud19200 | define | Serial |
| baud2400 | define | Serial |
| baud300 | define | Serial |
| baud3600 | define | Serial |
| baud4800 | define | Serial |
| baud57600 | define | Serial |
| baud600 | define | Serial |
| baud7200 | define | Serial |
| baud9600 | define | Serial |
| bdConv | define | Packages |
| bDevCitor | define | Printing |

| Identifier | Type | Manual page |
|--------------|----------|-------------|
| bDevLaser | define | Printing |
| bdNamErr | define | Errors |
| BDraftLoop | define | Printing |
| BDSElement | type | AppleTalk |
| BDSPtr | type | AppleTalk |
| BDSType | type | AppleTalk |
| BeginUpdate | function | Windows |
| BitAnd | function | ToolUtils |
| BitClr | function | ToolUtils |
| BitMap | type | Quickdraw |
| BitMapType | type | AppleTalk |
| BitNot | function | ToolUtils |
| BitOr | function | ToolUtils |
| Bits16 | type | Quickdraw |
| BitSet | function | ToolUtils |
| BitShift | function | ToolUtils |
| BitTst | function | ToolUtils |
| BitXor | function | ToolUtils |
| blackBit | define | Quickdraw |
| blackColor | define | Quickdraw |
| BlockMove | function | Memory |
| blueBit | define | Quickdraw |
| blueColor | define | Quickdraw |
| bold | define | Quickdraw |
| Boolean | type | Types |
| breakEvent | define | Serial |
| breakRecd | define | Errors |
| BringToFront | function | Windows |
| bSpoolLoop | define | Printing |
| btnCtrl | define | Dialogs |
| btnState | define | Events |
| buf2SmallErr | define | Errors |
| BUFSIZ | define | setbuf (C) |
| Button | function | Events |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|-----------------|----------|-------------|---------------|----------|-------------|
| buttonMsg | define | Devices | clearerr | macro | ferror (C) |
| c2pstr | function | Strings | ClearMenuBar | function | Menus |
| cairo | define | Fonts | Clip3D | function | Graf3D |
| calcCRgns | define | Controls | ClipAbove | function | Windows |
| CalcMask | function | Quickdraw | ClipRect | function | Quickdraw |
| CalcMenuSize | function | Menus | clkRdErr | define | Errors |
| CalcVis | function | Windows | clkWrErr | define | Errors |
| CalcVisBehind | function | Windows | close | function | close (C) |
| calloc | function | malloc (C) | CloseDeskAcc | function | Desk |
| cancelButton | define | Dialogs | CloseDialog | function | Dialogs |
| cantStepErr | define | Errors | CloseDriver | function | Devices |
| CautionAlert | function | Dialogs | ClosePicture | function | Quickdraw |
| cbNotFound | define | AppleTalk | ClosePoly | function | Quickdraw |
| ceil | function | floor (C) | ClosePort | function | Quickdraw |
| Cell | type | Lists | CloseResFile | function | Resources |
| century | define | Packages | CloseRgn | function | Quickdraw |
| cfree | function | malloc (C) | CloseWindow | function | Windows |
| ChangedResource | function | Resources | ClrAppFiles | function | SegLoad |
| charCodeMask | define | Events | cmdKey | define | Events |
| Chars | type | TextEdit | CMovePBRc | type | Files |
| CharsHandle | type | TextEdit | CntrlParam | type | Devices |
| CharsPtr | type | TextEdit | ColorBit | function | Quickdraw |
| CharWidth | function | Quickdraw | commandMark | define | Fonts |
| checkBoxProc | define | Controls | CompactMem | function | Memory |
| CheckItem | function | Menus | compound | function | SANE |
| checkMark | define | Fonts | condense | define | Quickdraw |
| CheckUpdate | function | Windows | Control | function | Devices |
| chkCtrl | define | Dialogs | controlErr | define | Errors |
| chooserID | define | Devices | ControlHandle | type | Controls |
| ckSumErr | define | Errors | ControlPtr | type | Controls |
| classcomp | function | SANE | ControlRecord | type | Controls |
| classdouble | function | SANE | CopyBits | function | Quickdraw |
| clasextended | function | SANE | CopyMask | function | Quickdraw |
| classfloat | function | SANE | CopyRgn | function | Quickdraw |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|-----------------|----------|-------------|----------------|----------|-------------|
| copySign | function | SANE | dataVerErr | define | Errors |
| corErr | define | Errors | Date2Secs | function | OSUtils |
| cos | function | trig(C) | DateForm | type | Packages |
| cosh | function | sinh(C) | DateTimeRec | type | OSUtils |
| CouldAlert | function | Dialogs | dayLdingZ | define | Packages |
| CouldDialog | function | Dialogs | DBLPRECISION | define | SANE |
| Count1Resources | function | Resources | dBoxProc | define | Windows |
| Count1Types | function | Resources | DctlEntry | type | Devices |
| CountAppFiles | function | SegLoad | DctlHandle | type | Devices |
| CountMItems | function | Menus | DctlPtr | type | Devices |
| CountResources | function | Resources | DDPCloseSocket | function | AppleTalk |
| CountTypes | function | Resources | ddpLenErr | define | Errors |
| courier | define | Fonts | DDPOpenSocket | function | AppleTalk |
| creat | function | creat(C) | ddpProto | literal | AppleTalk |
| Create | function | Files | DDPRdCancel | function | AppleTalk |
| CreateResFile | function | Resources | DDPRead | function | AppleTalk |
| crossCursor | define | ToolUtils | ddpSize | define | AppleTalk |
| ctaIcon | define | Dialogs | ddpSktErr | define | Errors |
| ctrlItem | define | Dialogs | DDPWrite | function | AppleTalk |
| ctsEvent | define | Serial | DDTMAC | define | SCSI |
| CurResFile | function | Resources | dec2num | function | SANE |
| currLeadingZ | define | Packages | dec2str | function | SANE |
| currNegSym | define | Packages | decform | type | SANE |
| currSymLead | define | Packages | decimal | type | SANE |
| currTrailingZ | define | Packages | DECSTROUTLEN | define | SANE |
| CursHandle | type | ToolUtils | Delay | function | OSUtils |
| Cursor | type | Quickdraw | DeleteMenu | function | Menus |
| CursPtr | type | ToolUtils | DelMenuItem | function | Menus |
| cyanBit | define | Quickdraw | DeltaPoint | function | ToolUtils |
| cyanColor | define | Quickdraw | DENORMALNUM | define | SANE |
| data5 | define | Serial | Dequeue | function | OSUtils |
| data6 | define | Serial | deSelect | define | Desk |
| data7 | define | Serial | deselectMsg | define | Devices |
| data8 | define | Serial | deskPatID | define | Windows |

| Identifier | Type | Manual page |
|----------------|----------|-------------|
| DetachResource | function | Resources |
| Device | function | Devices |
| dialogKind | define | Windows |
| DialogPeek | type | Dialogs |
| DialogPtr | type | Dialogs |
| DialogRecord | type | Dialogs |
| DialogSelect | function | Dialogs |
| DialogTemplate | type | Dialogs |
| DialogTHndl | type | Dialogs |
| DialogTPtr | type | Dialogs |
| diamondMark | define | Fonts |
| DIBadMount | function | Packages |
| DiffRgn | function | Quickdraw |
| DIFormat | function | Packages |
| DILoad | function | Packages |
| DInfo | type | Files |
| dInstErr | define | Errors |
| dirFulErr | define | Errors |
| DirInfo | type | Files |
| dirNFErr | define | Errors |
| DisableItem | function | Menus |
| DiskEject | function | disks |
| diskEvt | define | Events |
| diskMask | define | Events |
| dispCntl | define | Controls |
| DisposDialog | function | Dialogs |
| DisposeControl | function | Controls |
| DisposeMenu | function | Menus |
| DisposeRgn | function | Quickdraw |
| DisposeWindow | function | Windows |
| DisposHandle | function | Memory |
| DisposPtr | function | Memory |
| DIUnLoad | function | Packages |
| DIVBYZERO | define | SANE |

| Identifier | Type | Manual page |
|--------------|----------|-------------|
| DIVerify | function | Packages |
| DIZero | function | Packages |
| DlgCopy | function | Dialogs |
| DlgCut | function | Dialogs |
| DlgDelete | function | Dialogs |
| DlgPaste | function | Dialogs |
| dmy | define | Packages |
| documentProc | define | Windows |
| DOWNWARD | define | SANE |
| dragCntl | define | Controls |
| DragControl | function | Controls |
| DragGrayRgn | function | Windows |
| DragWindow | function | Windows |
| DrawlControl | function | Controls |
| DrawChar | function | Quickdraw |
| drawCntl | define | Controls |
| DrawControls | function | Controls |
| DrawDialog | function | Dialogs |
| DrawGrowIcon | function | Windows |
| DrawMenuBar | function | Menus |
| DrawNew | function | Windows |
| DrawPicture | function | Quickdraw |
| DrawString | function | Quickdraw |
| DrawText | function | Quickdraw |
| dRemoveErr | define | Errors |
| DriveKind | type | disks |
| driverEvt | define | Events |
| driverMask | define | Events |
| DriveStatus | function | disks |
| DrvQE1 | type | Files |
| DrvQE11 | type | Files |
| DrvQE1Ptr | type | Files |
| drvQType | literal | OSUtils |
| DrvStsHard20 | type | disks |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|--------------|----------|---------------|-------------|----------|---------------|
| DrvStsSory | type | disks | editText | define | Dialogs |
| dsAddressErr | define | Errors | EEXIST | define | errorintro(C) |
| dsBusErr | define | Errors | EINVAL | define | errorintro(C) |
| dsChkErr | define | Errors | EIO | define | errorintro(C) |
| dsCoreErr | define | Errors | EISDIR | define | errorintro(C) |
| dsFPErr | define | Errors | Eject | function | Files |
| dsFSErr | define | Errors | EMFILE | define | errorintro(C) |
| dsIllInstErr | define | Errors | EMLINK | define | errorintro(C) |
| dsIOCoreErr | define | Errors | EmptyHandle | function | Memory |
| dsIrqErr | define | Errors | EmptyRect | function | Quickdraw |
| dskFulErr | define | Errors | EmptyRgn | function | Quickdraw |
| dskInit | define | Packages | EnableItem | function | Menus |
| dsLineAErr | define | Errors | EndUpdate | function | Windows |
| dsLineFErr | define | Errors | ENFILE | define | errorintro(C) |
| dsLoadErr | define | Errors | ENODEV | define | errorintro(C) |
| dsMemFullErr | define | Errors | ENOENT | define | errorintro(C) |
| dsMiscErr | define | Errors | ENOMEM | define | errorintro(C) |
| dsNoPackErr | define | Errors | ENOSRRC | define | errorintro(C) |
| dsNotThe1 | define | Errors | ENOSPC | define | errorintro(C) |
| dsOvflowErr | define | Errors | ENOTDIR | define | errorintro(C) |
| dsPrivErr | define | Errors | Enqueue | function | OSUtils |
| dsReinsert | define | Errors | EntityName | type | AppleTalk |
| dsStKnHeap | define | Errors | EntityPtr | type | AppleTalk |
| dsSysErr | define | Errors | environment | type | SANE |
| dsTraceErr | define | Errors | Environs | function | OSUtils |
| dsZeroDivErr | define | Errors | ENXIO | define | errorintro(C) |
| dummyType | literal | OSUtils | EOF | define | stdio(C) |
| dup | function | dup(C) | eofErr | define | Errors |
| dupFNErr | define | Errors | EqualPt | function | Quickdraw |
| DXInfo | type | Files | EqualRect | function | Quickdraw |
| E2BIG | define | errorintro(C) | EqualRgn | function | Quickdraw |
| EACCESS | define | errorintro(C) | EqualString | function | OSUtils |
| EBADF | define | errorintro(C) | EQUALTO | define | SANE |
| ecvt | function | ecvt(C) | erase | literal | Quickdraw |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|----------------|----------|---------------|-------------|----------|-------------|
| EraseArc | function | Quickdraw | fclose | function | fclose(C) |
| EraseOval | function | Quickdraw | fcntl | function | fcntl(C) |
| ErasePoly | function | Quickdraw | FCPBRec | type | Files |
| EraseRect | function | Quickdraw | fcvt | function | ecvt(C) |
| EraseRgn | function | Quickdraw | F_DELETE | define | faccess(C) |
| EraseRoundRect | function | Quickdraw | fDesktop | define | Files |
| EROFS | define | errorintro(C) | fDisk | define | Files |
| errno | int | errorintro(C) | fdopen | function | fopen(C) |
| ErrorSound | function | Dialogs | F_DUPFD | define | fcntl(C) |
| ESPIPE | define | errorintro(C) | feedCut | literal | Printing |
| evenParity | define | Serial | feedFanfold | literal | Printing |
| EventAvail | function | Events | feedMechCut | literal | Printing |
| EventRecord | type | Events | feedOther | literal | Printing |
| everyEvent | define | Events | feof | macro | ferror(C) |
| EvQEL | type | OSEvents | ferror | macro | ferror(C) |
| evtNotEnb | define | Errors | fflush | function | fclose(C) |
| evType | literal | OSUtils | ffMode | define | Sound |
| exception | type | SANE | FFSynthRec | type | Sound |
| excessCollsns | define | Errors | fgetc | function | getc(C) |
| exit | function | exit(C) | fgets | function | gets(C) |
| _exit | function | exit(C) | F_FONTINFO | define | faccess(C) |
| ExitToShell | function | SegLoad | F_GPRINTREC | define | faccess(C) |
| exp | function | exp(C) | F_GTABINFO | define | faccess(C) |
| exp1 | function | SANE | fHasBundle | define | Files |
| exp2 | function | SANE | FILE | type | stdio(C) |
| expand | define | Quickdraw | fileno | macro | ferror(C) |
| extFSErr | define | Errors | FileParam | type | Files |
| EXTPRECISION | define | SANE | fill | literal | Quickdraw |
| extractErr | define | Errors | FillArc | function | Quickdraw |
| fabs | function | floor(C) | fillList | define | Desk |
| faccess | function | faccess(C) | fillListMsg | define | Devices |
| false | literal | Types | FillOval | function | Quickdraw |
| FamRec | type | Fonts | FillPoly | function | Quickdraw |
| fBsyErr | define | Errors | FillRect | function | Quickdraw |

| Identifier | Type | Manual page |
|----------------|----------|-------------|
| FillRgn | function | Quickdraw |
| FillRoundRect | function | Quickdraw |
| FindControl | function | Controls |
| FindDItem | function | Dialogs |
| FindWindow | function | Windows |
| FInfo | type | Files |
| FInitQueue | function | Files |
| fInvisible | define | Files |
| FIOBUFSIZE | define | ioctl(C) |
| FIODUPFD | define | ioctl(C) |
| FIOFNAME | define | ioctl(C) |
| FIOINTERACTIVE | define | ioctl(C) |
| FIOLSEEK | define | ioctl(C) |
| FIOREFNUM | define | ioctl(C) |
| FIOSETEOF | define | ioctl(C) |
| firstDskErr | define | Errors |
| Fix2Frac | function | FixMath |
| Fix2Long | function | FixMath |
| Fix2X | function | FixMath |
| FixAtan2 | function | FixMath |
| FixDiv | function | FixMath |
| Fixed | type | Types |
| FIXEDDECIMAL | define | SANE |
| fixedFont | define | Fonts |
| FixMul | function | ToolUtils |
| FixRatio | function | ToolUtils |
| FixRound | function | ToolUtils |
| FlashMenuBar | function | Menus |
| fLckdErr | define | Errors |
| FLOATDECIMAL | define | SANE |
| FLCATPRECISION | define | SANE |
| floor | function | floor(C) |
| flPoint | define | Packages |
| FlushEvents | function | OSEvents |

| Identifier | Type | Manual page |
|----------------|----------|-------------|
| FlushVol | function | Files |
| FMInput | type | Fonts |
| fmod | function | floor(C) |
| FMOutPtr | type | Fonts |
| FMOutput | type | Fonts |
| FmtDefaults | define | DiskInit |
| fnfErr | define | Errors |
| fnOpnErr | define | Errors |
| fOnDesk | define | Files |
| fontDecError | define | Errors |
| FontInfo | type | Quickdraw |
| FontRec | type | Fonts |
| fontSubErr | define | Errors |
| fontWid | define | Fonts |
| F_OPEN | define | faccess(C) |
| fopen | function | fopen(C) |
| ForeColor | function | Quickdraw |
| fprintf | function | printf(C) |
| fputc | function | putc(C) |
| fputs | function | puts(C) |
| Frac2Fix | function | FixMath |
| Frac2X | function | FixMath |
| FracCos | function | FixMath |
| FracDiv | function | FixMath |
| FracMul | function | FixMath |
| FracSin | function | FixMath |
| FracSqrt | function | FixMath |
| frame | literal | Quickdraw |
| FrameArc | function | Quickdraw |
| FrameOval | function | Quickdraw |
| FramePoly | function | Quickdraw |
| FrameRect | function | Quickdraw |
| FrameRgn | function | Quickdraw |
| FrameRoundRect | function | Quickdraw |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|--------------|----------|-------------|-------------------|----------|-------------|
| framingErr | define | Errors | FSWrite | function | Files |
| fread | function | fread(C) | fsWrPerm | define | Files |
| free | function | malloc(C) | ftell | function | fseek(C) |
| FreeAlert | function | Dialogs | ftMode | define | Sound |
| FreeDialog | function | Dialogs | fTrash | define | Files |
| FreeMem | function | Memory | fwrite | function | fread(C) |
| FreeWave | type | Sound | fxdFntH | define | Fonts |
| F_RENAME | define | faccess(C) | fxdFntHW | define | Fonts |
| freopen | function | fopen(C) | fxdFntW | define | Fonts |
| frexp | function | frexp(C) | FXInfo | type | Files |
| FrontWindow | function | Windows | geneva | define | Fonts |
| fsAtMark | define | disks | Get1IndResource | function | Resources |
| fsAtMark | define | Files | Get1IndType | function | Resources |
| fscanf | function | scanf(C) | Get1NamedResource | function | Resources |
| FSClose | function | Files | Get1Resource | function | Resources |
| fsCurPerm | define | Files | GetAlrtStage | function | Dialogs |
| FSDelete | function | Files | GetAppFiles | function | SegLoad |
| fsDSIntErr | define | Errors | GetAppLimit | function | Memory |
| fseek | function | fseek(C) | GetAppParms | function | SegLoad |
| F_SFONINFO | define | faccess(C) | getc | macro | getc(C) |
| fsFromLEOF | define | Files | getCancel | define | Packages |
| fsFromMark | define | disks | GetCaretTime | function | Events |
| fsFromMark | define | Files | getchar | macro | getc(C) |
| fsFromStart | define | disks | GetClip | function | Quickdraw |
| fsFromStart | define | Files | GetCRefCon | function | Controls |
| FSOpen | function | Files | GetCTitle | function | Controls |
| F_SPRINTF | define | faccess(C) | GetCtlAction | function | Controls |
| fsQType | literal | OSUtils | GetCtlMax | function | Controls |
| fsRdPerm | define | Files | GetCtlMin | function | Controls |
| fsRdWrPerm | define | Files | GetCtlValue | function | Controls |
| fsRdWrShPerm | define | Files | GetCursor | function | ToolUtils |
| FSRead | function | Files | GetDateTime | function | OSUtils |
| fsRnErr | define | Errors | GetDb1Time | function | Events |
| F_STABINFO | define | faccess(C) | GetDCTLEntry | function | Devices |

| Identifier | Type | Manual page |
|------------------|----------|-------------|
| GetDItem | function | Dialogs |
| getDlgID | define | Packages |
| getDrive | define | Packages |
| GetDrvQHdr | function | Files |
| getEject | define | Packages |
| getenv | function | getenv(C) |
| getenvironment | function | SANE |
| GetEOF | function | Files |
| GetEvQHdr | function | OSEvents |
| GetFInfo | function | Files |
| GetFontInfo | function | Quickdraw |
| GetFPos | function | Files |
| GetFSQHdr | function | Files |
| gethaltvector | function | SANE |
| GetHandleSize | function | Memory |
| GetIcon | function | ToolUtils |
| GetIndPattern | function | ToolUtils |
| GetIndResource | function | Resources |
| GetIndString | function | ToolUtils |
| GetIndType | function | Resources |
| GetItem | function | Menus |
| GetItemIcon | function | Menus |
| GetItemMark | function | Menus |
| GetItemStyle | function | Menus |
| GetIText | function | Dialogs |
| GetKeys | function | Events |
| GetMenu | function | Menus |
| GetMenuBar | function | Menus |
| GetMHandle | function | Menus |
| GetMouse | function | Events |
| GetNamedResource | function | Resources |
| GetNewControl | function | Controls |
| GetNewDialog | function | Dialogs |
| GetNewMBar | function | Menus |

| Identifier | Type | Manual page |
|-----------------|----------|-------------|
| GetNewWindow | function | Windows |
| GetNextEvent | function | Events |
| getNmList | define | Packages |
| GetNodeAddress | function | AppleTalk |
| getOpen | define | Packages |
| GetOSEvent | function | OSEvents |
| GetPattern | function | ToolUtils |
| GetPen | function | Quickdraw |
| GetPenState | function | Quickdraw |
| GetPicture | function | ToolUtils |
| GetPixel | function | Quickdraw |
| GetPort | function | Quickdraw |
| GetPort3D | function | Graf3D |
| getprecision | function | SANE |
| GetPtrSize | function | Memory |
| GetResAttrs | function | Resources |
| GetResFileAttrs | function | Resources |
| GetResInfo | function | Resources |
| GetResource | function | Resources |
| getround | function | SANE |
| gets | function | gets(C) |
| GetScrap | function | Scrap |
| getScroll | define | Packages |
| getSel | define | Desk |
| getSelMsg | define | Devices |
| GetString | function | ToolUtils |
| GetSysPPtr | function | OSUtils |
| GetTime | function | OSUtils |
| GetTrapAddress | function | OSUtils |
| GetVBLQHdr | function | Retrace |
| GetVCBQHdr | function | Files |
| GetVInfo | function | Files |
| GetVol | function | Files |
| GetVRefNum | function | Files |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|---------------|----------|-------------|---------------|----------|-------------|
| getw | function | getc(C) | HidePen | function | Quickdraw |
| GetWindowPic | function | Windows | HideWindow | function | Windows |
| GetWMgrPort | function | Windows | HiliteControl | function | Controls |
| GetWRefCon | function | Windows | HiliteMenu | function | Menus |
| GetWTitle | function | Windows | HiliteWindow | function | Windows |
| GetZone | function | Memory | HIOParam | type | Files |
| gfpErr | define | Errors | HiWord | function | ToolUtils |
| GlobalToLocal | function | Quickdraw | HLock | function | Memory |
| GrafDevice | function | Quickdraw | HNoPurge | function | Memory |
| GrafPort | type | Quickdraw | HomeResFile | function | Resources |
| GrafPtr | type | Quickdraw | HPurge | function | Memory |
| GrafVerb | type | Quickdraw | hrLeadingZ | define | Packages |
| GREATERTHAN | define | SANE | HSetRBit | function | Memory |
| greenBit | define | Quickdraw | HSetState | function | Memory |
| greenColor | define | Quickdraw | HUnlock | function | Memory |
| GrowWindow | function | Windows | HVolumeParam | type | Files |
| GZSaveHnd | function | Memory | hwOverrunErr | define | Errors |
| haltvector | type | SANE | hypot | function | hypot(C) |
| HandAndHand | function | OSUtils | iBeamCursor | define | ToolUtils |
| Handle | type | Types | iconItem | define | Dialogs |
| HandleZone | function | Memory | Identity | function | Graf3D |
| HandToHand | function | OSUtils | iIOAbort | define | Printing |
| hard20 | literal | disks | iMemFullErr | define | Printing |
| hAxisOnly | define | Controls | inButton | define | Controls |
| hAxisOnly | define | Windows | inCheckbox | define | Controls |
| HClrRBit | function | Memory | inContent | define | Windows |
| helvetica | define | Fonts | inDesk | define | Windows |
| HFileInfo | type | Files | index | function | string(C) |
| HFileParam | type | Files | inDownButton | define | Controls |
| HFSDefaults | type | DiskInit | inDrag | define | Windows |
| HGetState | function | Memory | INEXACT | define | SANE |
| HideControl | function | Controls | inf | function | SANE |
| HideCursor | function | Quickdraw | INFINITE | define | SANE |
| HideDItem | function | Dialogs | InfoScrap | function | Scrap |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|---------------|----------|-------------|-----------------|----------|-------------|
| inGoAway | define | Windows | Int11Rec | type | Packages |
| inGrow | define | Windows | intUtil | define | Packages |
| InitAllPacks | function | Packages | inUpButton | define | Controls |
| InitApplZone | function | Memory | INVALID | define | SANE |
| initCntl | define | Controls | InvalRect | function | Windows |
| InitCursor | function | Quickdraw | InvalRgn | function | Windows |
| InitDialogs | function | Dialogs | inverseBit | define | Quickdraw |
| InitGraf | function | Quickdraw | invert | literal | Quickdraw |
| InitGrf3D | function | Graf3D | InvertArc | function | Quickdraw |
| initIWMErr | define | Errors | InvertOval | function | Quickdraw |
| InitMenus | function | Menus | InvertPoly | function | Quickdraw |
| InitPack | function | Packages | InvertRect | function | Quickdraw |
| InitPort | function | Quickdraw | InvertRgn | function | Quickdraw |
| InitResources | function | Resources | InvertRoundRect | function | Quickdraw |
| InitUtil | function | OSUtils | inZoomIn | define | Windows |
| InitWindows | function | Windows | inZoomOut | define | Windows |
| InitZone | function | Memory | ioctl | function | ioctl(C) |
| inMenuBar | define | Windows | ioErr | define | Errors |
| inPageDown | define | Controls | _IOFBF | define | setbuf(C) |
| inPageUp | define | Controls | _IOLBF | define | setbuf(C) |
| InsertMenu | function | Menus | _IONBF | define | setbuf(C) |
| InsertResMenu | function | Menus | IOParm | type | Files |
| InsetRect | function | Quickdraw | ioQType | literal | OSUtils |
| InsetRgn | function | Quickdraw | iPFMaxPgs | define | Printing |
| InsMenuItem | function | Menus | ipower | function | SANE |
| InsTime | function | Time | iPrAbort | define | Printing |
| inSysWindow | define | Windows | iPrBitsCtl | define | Printing |
| Int64Bit | type | ToolUtils | iPrDevCtl | define | Printing |
| inThumb | define | Controls | iPrDrvRef | define | Printing |
| Int10Hndl | type | Packages | iPrIOCtl | define | Printing |
| Int10Ptr | type | Packages | iPrPgFract | define | Printing |
| Int10Rec | type | Packages | iPrSavPFil | define | Printing |
| Int11Hndl | type | Packages | isalnum | macro | ctype(C) |
| Int11Ptr | type | Packages | isalpha | macro | ctype(C) |

| Identifier | Type | Manual page |
|---------------|----------|-------------|
| isascii | macro | ctype(C) |
| IsATPOpen | function | AppleTalk |
| iscntrl | macro | ctype(C) |
| IsDialogEvent | function | Dialogs |
| isdigit | macro | ctype(C) |
| isgraph | macro | ctype(C) |
| islower | macro | ctype(C) |
| IsMPPPOpen | function | AppleTalk |
| isprint | macro | ctype(C) |
| ispunct | macro | ctype(C) |
| isspace | macro | ctype(C) |
| isupper | macro | ctype(C) |
| isxdigit | macro | ctype(C) |
| italic | define | Quickdraw |
| itemDisable | define | Dialogs |
| IUCompString | function | Packages |
| IUDatePString | function | Packages |
| IUDateString | function | Packages |
| IUEqualString | function | Packages |
| IUGetInt1 | function | Packages |
| IUMagIDString | function | Packages |
| IUMagString | function | Packages |
| IUMetric | function | Packages |
| IUSetInt1 | function | Packages |
| IUTimePString | function | Packages |
| IUTimeString | function | Packages |
| jmp_buf | type | setjmp(C) |
| keyCodeMask | define | Events |
| keyDown | define | Events |
| keyDownMask | define | Events |
| keyMap | type | Events |
| keyUp | define | Events |
| keyUpMask | define | Events |
| KillControls | function | Controls |

| Identifier | Type | Manual page |
|------------------|----------|-------------|
| KillIO | function | Devices |
| KillPicture | function | Quickdraw |
| KillPoly | function | Quickdraw |
| LActivate | function | Lists |
| LAddColumn | function | Lists |
| LAddRow | function | Lists |
| LAddToCell | function | Lists |
| LAPAdrBlock | type | AppleTalk |
| LAPCloseProtocol | function | AppleTalk |
| LAPOpenProtocol | function | AppleTalk |
| LAPProtErr | define | Errors |
| lapProto | literal | AppleTalk |
| LAPRead | function | AppleTalk |
| lapSize | define | AppleTalk |
| LAPWrite | function | AppleTalk |
| lastDskErr | define | Errors |
| LAutoScroll | function | Lists |
| LCellSize | function | Lists |
| LClick | function | Lists |
| LCloseMsg | define | Lists |
| LClrCell | function | Lists |
| LDelColumn | function | Lists |
| LDelRow | function | Lists |
| ldexp | function | frexp(C) |
| LDispose | function | Lists |
| LDoDraw | function | Lists |
| LDoHAutoscroll | define | Lists |
| LDoVAutoscroll | define | Lists |
| LDraw | function | Lists |
| LDrawMsg | define | Lists |
| LESSTHAN | define | SANE |
| lExtendDrag | define | Lists |
| LFind | function | Lists |
| LGetCell | function | Lists |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|---------------|----------|-------------|--------------|----------|---------------|
| LGetSelect | function | Lists | lOnlyOne | define | Lists |
| LHiliteMsg | define | Lists | LookAt | function | Graf3D |
| Lire | function | Quickdraw | losAngeles | define | Fonts |
| Line2D | function | Graf3D | LoWord | function | ToolUtils |
| Line3D | function | Graf3D | lPaintBits | define | Printing |
| LineTo | function | Quickdraw | lPrLFSixth | define | Printing |
| LineTo2D | function | Graf3D | lPrLineFeed | define | Printing |
| LineTo3D | function | Graf3D | lPrPageEnd | define | Printing |
| LInitMsg | define | Lists | lPrReset | define | Printing |
| ListHandle | type | Lists | LRect | function | Lists |
| ListMgr | define | Packages | lScreenBits | define | Printing |
| ListPtr | type | Lists | LScroll | function | Lists |
| ListRec | type | Lists | LSearch | function | Lists |
| LLastClick | function | Lists | lseek | function | lseek(C) |
| LNew | function | Lists | LSetCell | function | Lists |
| LNextCell | function | Lists | LSetSelect | function | Lists |
| LNoDisjoint | define | Lists | LSize | function | Lists |
| LNoExtend | define | Lists | LUpdate | function | Lists |
| LNoNilHilite | define | Lists | lUseSense | define | Lists |
| LNoRect | define | Lists | macMachine | define | OSUtils |
| LoadResource | function | Resources | MacOSErr | short | errorintro(C) |
| LoadScrap | function | Scrap | macXLMachine | define | OSUtils |
| LocalToGlobal | function | Quickdraw | magentaBit | define | Quickdraw |
| log | function | exp(C) | magentaColor | define | Quickdraw |
| log1 | function | SANE | malloc | function | malloc(C) |
| log10 | function | exp(C) | mapChanged | define | Resources |
| log2 | function | SANE | mapCompact | define | Resources |
| logo | function | SANE | MapPoly | function | Quickdraw |
| london | define | Fonts | MapPt | function | Quickdraw |
| long | type | Types | mapReadErr | define | Errors |
| Long2Fix | function | FixMath | mapReadOnly | define | Resources |
| longDate | literal | Packages | MapRect | function | Quickdraw |
| longjmp | function | setjmp(C) | MapRgn | function | Quickdraw |
| LongMul | function | ToolUtils | MaxApplZone | function | Memory |

| Identifier | Type | Manual page |
|--------------|----------|-------------|
| MaxBlock | function | Memory |
| MaxMem | function | Memory |
| maxSize | define | Memory |
| MaxSizeRsrc | function | Resources |
| mChooseMsg | define | Menus |
| mDownMask | define | Events |
| mDrawMsg | define | Menus |
| mdy | define | Packages |
| MeasureText | function | Quickdraw |
| memAdrErr | define | Errors |
| memAZErr | define | Errors |
| memBCErr | define | Errors |
| memcpy | function | Memory (C) |
| memchr | function | Memory (C) |
| memcmp | function | Memory (C) |
| memcpy | function | Memory (C) |
| MemError | function | Memory |
| memFullErr | define | Errors |
| memLockedErr | define | Errors |
| memPCErr | define | Errors |
| memPurErr | define | Errors |
| memROZErr | define | Errors |
| memSCErr | define | Errors |
| memset | function | Memory (C) |
| memWZErr | define | Errors |
| MenuHandle | type | Menus |
| MenuInfo | type | Menus |
| MenuKey | function | Menus |
| menuPrgErr | define | Errors |
| MenuPtr | type | Menus |
| MenuSelect | function | Menus |
| minLeadingZ | define | Packages |
| minLdingZ | define | Packages |
| mobile | define | Fonts |

| Identifier | Type | Manual page |
|--------------|----------|-------------|
| ModalDialog | function | Dialogs |
| modf | function | frexp(C) |
| monaco | define | Fonts |
| MoreMasters | function | Memory |
| mouseDown | define | Events |
| mouseUp | define | Events |
| Move | function | Quickdraw |
| Move2D | function | Graf3D |
| Move3D | function | Graf3D |
| MoveControl | function | Controls |
| MoveHHi | function | Memory |
| MovePortTo | function | Quickdraw |
| MoveTo | function | Quickdraw |
| MoveTo2D | function | Graf3D |
| MoveTo3D | function | Graf3D |
| MoveWindow | function | Windows |
| MPPClose | function | AppleTalk |
| MPPOpen | function | AppleTalk |
| mSizeMsg | define | Menus |
| Munger | function | ToolUtils |
| mUpMask | define | Events |
| MyAction | function | Controls |
| MyAction | function | Windows |
| MyArc | function | Quickdraw |
| MyBits | function | Quickdraw |
| MyClikLoop | function | TextEdit |
| MyComment | function | Quickdraw |
| MyControl | function | Controls |
| MyDlg | function | Packages |
| MyFileFilter | function | Packages |
| MyFilter | function | Dialogs |
| MyGetPic | function | Quickdraw |
| MyGrowZone | function | Memory |
| MyItem | function | Dialogs |

| Identifier | Type | Manual page |
|--------------|----------|-------------|
| MyLine | function | Quickdraw |
| MyListDef | function | Lists |
| MyMenu | function | Menus |
| MyOval | function | Quickdraw |
| MyPoly | function | Quickdraw |
| MyPutPic | function | Quickdraw |
| MyRect | function | Quickdraw |
| MyRgn | function | Quickdraw |
| MyRRect | function | Quickdraw |
| MySound | function | Dialogs |
| MyText | function | Quickdraw |
| MyTxMeas | function | Quickdraw |
| MyWindow | function | Windows |
| MyWordBreak | function | TextEdit |
| nan | function | SANE |
| nbpBuffOvr | define | AppleTalk |
| nbpConfDiff | define | AppleTalk |
| NBPConfirm | function | AppleTalk |
| nbpDuplicate | define | AppleTalk |
| NBPExtract | function | AppleTalk |
| NBPLoad | function | AppleTalk |
| NBPLookUp | function | AppleTalk |
| nbpNISErr | define | Errors |
| nbpNoConfirm | define | AppleTalk |
| nbpNotFound | define | AppleTalk |
| nbpProto | literal | AppleTalk |
| NBPRegister | function | AppleTalk |
| NBPRemove | function | AppleTalk |
| nbpSize | define | AppleTalk |
| NBPUnLoad | function | AppleTalk |
| networkEvt | define | Events |
| networkMask | define | Events |
| NewControl | function | Controls |
| NewDialog | function | Dialogs |

| Identifier | Type | Manual page |
|-----------------|----------|-------------|
| NewEmptyHandle | function | Memory |
| NewHandle | function | Memory |
| NewMenu | function | Menus |
| NewPtr | function | Memory |
| NewRgn | function | Quickdraw |
| newSelMsg | define | Devices |
| NewString | function | ToolUtils |
| NewWindow | function | Windows |
| newYork | define | Fonts |
| nextdouble | function | SANE |
| nextextended | function | SANE |
| nextfloat | function | SANE |
| NGetTrapAddress | function | OSUtils |
| nil | define | Types |
| nilHandleErr | define | Errors |
| noAdmMkErr | define | Errors |
| noBridgeErr | define | Errors |
| noConstraint | define | Controls |
| noConstraint | define | Windows |
| noDataArea | define | AppleTalk |
| noDriveErr | define | Errors |
| noDtaMkErr | define | Errors |
| noErr | define | Errors |
| noGrowDocProc | define | Windows |
| noMacDskErr | define | Errors |
| noMark | define | Menus |
| noMPPErr | define | Errors |
| noNybErr | define | Errors |
| noParity | define | Serial |
| noRelErr | define | Errors |
| normal | define | Quickdraw |
| normalBit | define | Quickdraw |
| NORMALNUM | define | SANE |
| noScrapErr | define | Errors |

| Identifier | Type | Manual page |
|------------------|----------|-------------|
| noSendResp | define | AppleTalk |
| NoteAlert | function | Dialogs |
| noteIcon | define | Dialogs |
| notOpenErr | define | Errors |
| notPatBic | define | Quickdraw |
| notPatCopy | define | Quickdraw |
| notPatOr | define | Quickdraw |
| notPatXor | define | Quickdraw |
| notSrcBic | define | Quickdraw |
| notSrcCopy | define | Quickdraw |
| notSrcOr | define | Quickdraw |
| notSrcXor | define | Quickdraw |
| noTypeErr | define | Errors |
| nsDrvErr | define | Errors |
| NSSetTrapAddress | function | OSUtils |
| nsvErr | define | Errors |
| NULL | define | stdio(C) |
| NULL | define | Types |
| nullEvent | define | Events |
| num2dec | function | SANE |
| numclass | type | SANE |
| NumToString | function | Packages |
| O_APPEND | define | open(C) |
| ObscureCursor | function | Quickdraw |
| O_CREAT | define | open(C) |
| oddParity | define | Serial |
| O_EXCL | define | open(C) |
| offLinErr | define | Errors |
| OffsetPoly | function | Quickdraw |
| OffsetRect | function | Quickdraw |
| OffsetRgn | function | Quickdraw |
| okButton | define | Dialogs |
| onexit | function | onexit(C) |
| open | function | open(C) |

| Identifier | Type | Manual page |
|--------------|----------|-------------|
| Open3DPort | function | Graf3D |
| OpenDeskAcc | function | Desk |
| OpenDriver | function | Devices |
| openErr | define | Errors |
| OpenFRPerm | function | Resources |
| OpenPicture | function | Quickdraw |
| OpenPoly | function | Quickdraw |
| OpenPort | function | Quickdraw |
| OpenResFile | function | Resources |
| OpenRF | function | Files |
| openRFPerm | function | Resources |
| OpenRgn | function | Quickdraw |
| optionKey | define | Events |
| opWrErr | define | Errors |
| O_RDONLY | define | open(C) |
| O_RDWR | define | open(C) |
| O_RSRC | define | open(C) |
| OSErr | type | Types |
| OSEventAvail | function | OSEvents |
| OSTrap | literal | OSUtils |
| OSType | type | Types |
| O_TRUNC | define | open(C) |
| outline | define | Quickdraw |
| OVERFLOW | define | SANE |
| O_WRONLY | define | open(C) |
| p2cstr | function | Strings |
| PackBits | function | ToolUtils |
| paint | literal | Quickdraw |
| PaintArc | function | Quickdraw |
| PaintBehind | function | Windows |
| PaintOne | function | Windows |
| PaintOval | function | Quickdraw |
| PaintPoly | function | Quickdraw |
| PaintRect | function | Quickdraw |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|----------------|----------|-------------|---------------|----------|-------------|
| PaintRgn | function | Quickdraw | PBHGetFInfo | function | Files |
| PaintRoundRect | function | Quickdraw | PBHGetVInfo | function | Files |
| paramErr | define | Errors | PBHGetVol | function | Files |
| ParamText | function | Dialogs | PBHOpen | function | Files |
| parityErr | define | Errors | PBHOpenRF | function | Files |
| patBic | define | Quickdraw | PBHRename | function | Files |
| patCopy | define | Quickdraw | PBHRstFLock | function | Files |
| patHandle | type | ToolUtils | PBHSetFInfo | function | Files |
| patOr | define | Quickdraw | PBHSetFLock | function | Files |
| PatPtr | type | ToolUtils | PBHSetVol | function | Files |
| Pattern | type | Quickdraw | PBKillIO | function | Devices |
| patXor | define | Quickdraw | PBLockRange | function | Files |
| PBAllocate | function | Files | PBMountVol | function | Files |
| PBAllocContig | function | Files | PBOffline | function | Files |
| PBCatMove | function | Files | PBOpen | function | Files |
| PBClose | function | Files | PBOpenRF | function | Files |
| PBCloseWD | function | Files | PBOpenWD | function | Files |
| PBControl | function | Devices | PBRead | function | Files |
| PBCreate | function | Files | PBRename | function | Files |
| PBDelete | function | Files | PBRstFLock | function | Files |
| PBDirCreate | function | Files | PBSetCatInfo | function | Files |
| PBEject | function | Files | PBSetEOF | function | Files |
| PBFlushFile | function | Files | PBSetFInfo | function | Files |
| PBFlushVol | function | Files | PBSetFLock | function | Files |
| PBGetCatInfo | function | Files | PBSetFPos | function | Files |
| PBGetEOF | function | Files | PBSetFVers | function | Files |
| PBGetFCBInfo | function | Files | PBSetVInfo | function | Files |
| PBGetFInfo | function | Files | PBSetVol | function | Files |
| PBGetFPos | function | Files | PBStatus | function | Devices |
| PBGetVInfo | function | Files | PBUnLockRange | function | Files |
| PBGetVol | function | Files | PBUnmountVol | function | Files |
| PBGetWDInfo | function | Files | PBWrite | function | Files |
| PBHCreate | function | Files | PDSIGWORD | define | SCSI |
| PBHDelete | function | Files | PenMode | function | Quickdraw |

| Identifier | Type | Manual page |
|------------|----------|-------------|
| PenNormal | function | Quickdraw |
| PenPat | function | Quickdraw |
| PenSize | function | Quickdraw |
| PenState | type | Quickdraw |
| permErr | define | Errors |
| pi | function | SANE |
| PicComment | function | Quickdraw |
| PicHandle | type | Quickdraw |
| picItem | define | Dialogs |
| picLParen | define | Quickdraw |
| PicPtr | type | Quickdraw |
| picRParen | define | Quickdraw |
| Picture | type | Quickdraw |
| PinRect | function | Windows |
| Pitch | function | Graf3D |
| plainDBox | define | Windows |
| PlotIcon | function | ToolUtils |
| plusCursor | define | ToolUtils |
| Point | type | Types |
| Point2D | type | Graf3D |
| Point3D | type | Graf3D |
| Polygon | type | Quickdraw |
| PolyHandle | type | Quickdraw |
| PolyPtr | type | Quickdraw |
| Port3D | type | Graf3D |
| Port3DPtr | type | Graf3D |
| portInUse | define | Errors |
| portNotCf | define | Errors |
| PortSize | function | Quickdraw |
| posCntl | define | Controls |
| posErr | define | Errors |
| PostEvent | function | OSEvents |
| pow | function | exp(C) |
| power | function | SANE |

| Identifier | Type | Manual page |
|--------------|----------|-------------|
| PPostEvent | function | OSEvents |
| PrClose | function | Printing |
| PrCloseDoc | function | Printing |
| PrClosePage | function | Printing |
| PrCtlCall | function | Printing |
| PrDrvrClose | function | Printing |
| PrDrvrDCE | function | Printing |
| PrDrvrOpen | function | Printing |
| PrDrvrVers | function | Printing |
| PrError | function | Printing |
| PrimeTime | function | Time |
| prInitErr | define | Errors |
| PrintDefault | function | Printing |
| printf | function | printf(C) |
| PrJobDialog | function | Printing |
| PrJobMerge | function | Printing |
| procentry | function | SANE |
| procexit | function | SANE |
| ProcHandle | type | Types |
| ProcPtr | type | Types |
| PrOpen | function | Printing |
| PrOpenDoc | function | Printing |
| PrOpenPage | function | Printing |
| propFont | define | Fonts |
| prpFntH | define | Fonts |
| prpFntHW | define | Fonts |
| prpFntW | define | Fonts |
| PrPicFile | function | Printing |
| PrSetError | function | Printing |
| PrStlDialog | function | Printing |
| PrValidate | function | Printing |
| prWrErr | define | Errors |
| PScrapStuff | type | Scrap |
| Pt2Rect | function | Quickdraw |

| Identifier | Type | Manual page |
|-------------|----------|-------------|
| PtInRect | function | Quickdraw |
| PtInRgn | function | Quickdraw |
| Ptr | type | Types |
| PtrAndHand | function | OSUtils |
| PtrToHand | function | OSUtils |
| PtrToXHand | function | OSUtils |
| PtrZone | function | Memory |
| PtToAngle | function | Quickdraw |
| PurgeMem | function | Memory |
| PurgeSpace | function | Memory |
| pushButProc | define | Controls |
| putc | macro | putc(C) |
| putCancel | define | Packages |
| putchar | macro | putc(C) |
| putDlgID | define | Packages |
| putDrive | define | Packages |
| putEject | define | Packages |
| putName | define | Packages |
| puts | function | puts(C) |
| putSave | define | Packages |
| PutScrap | function | Scrap |
| putw | function | putc(C) |
| QDProcs | type | Quickdraw |
| QDProcsPtr | type | Quickdraw |
| QElem | type | OSUtils |
| QElemPtr | type | OSUtils |
| qErr | define | Errors |
| QHdr | type | OSUtils |
| QHdrPtr | type | OSUtils |
| QNAN | define | SANE |
| qsort | function | qsort(C) |
| QTypes | type | OSUtils |
| radConst | define | Graf3D |
| radCtrl | define | Dialogs |

| Identifier | Type | Manual page |
|-----------------|----------|-------------|
| radioButProc | define | Controls |
| RAMSDClose | function | Serial |
| RAMSOpen | function | Serial |
| rand | function | rand(C) |
| Random | function | Quickdraw |
| randomx | function | SANE |
| rcvrErr | define | Errors |
| rDocProc | define | Windows |
| rdVerify | define | disks |
| rdVerify | define | Files |
| read | function | read(C) |
| ReadDateTime | function | OSUtils |
| readErr | define | Errors |
| readQErr | define | Errors |
| realloc | function | malloc(C) |
| ReallocHandle | function | Memory |
| recNotFnd | define | AppleTalk |
| RecoverHandle | function | Memory |
| Rect | type | Types |
| RectInRgn | function | Quickdraw |
| RectRgn | function | Quickdraw |
| redBit | define | Quickdraw |
| redColor | define | Quickdraw |
| Region | type | Quickdraw |
| relation | function | SANE |
| ReleaseResource | function | Resources |
| relop | type | SANE |
| RelString | function | OSUtils |
| remainder | function | SANE |
| RemoveHdlBlks | function | AppleTalk |
| Rename | function | Files |
| reqAborted | define | AppleTalk |
| reqFailed | define | AppleTalk |
| resAttrErr | define | Errors |

| Identifier | Type | Manual page |
|----------------|----------|-------------|
| resChanged | define | Resources |
| resCtrl | define | Dialogs |
| ResError | function | Resources |
| ResetAlrtStage | function | Dialogs |
| resFNotFound | define | Errors |
| resLocked | define | Resources |
| resNotFound | define | Errors |
| resPreload | define | Resources |
| resProtected | define | Resources |
| resPurgeable | define | Resources |
| ResrvMem | function | Memory |
| resSysHeap | define | Resources |
| Restart | function | OSUtils |
| ResType | type | Types |
| RetransType | type | AppleTalk |
| rewind | function | fseek(C) |
| rfNumErr | define | Errors |
| RgnHandle | type | Quickdraw |
| RgnPtr | type | Quickdraw |
| rindex | function | string(C) |
| rint | function | SANE |
| RmveResource | function | Resources |
| rmvRefFailed | define | Errors |
| rmvResFailed | define | Errors |
| RmvTime | function | Time |
| Roll | function | Graf3D |
| rounddir | type | SANE |
| roundpre | type | SANE |
| RsrcMapEntry | function | Resources |
| RsrcZoneInit | function | Resources |
| RstFlock | function | Files |
| sanFran | define | Fonts |
| SaveOld | function | Windows |
| SBSIGWORD | define | SCSI |

| Identifier | Type | Manual page |
|---------------|----------|-------------|
| scAdd | define | SCSI |
| scalb | function | SANE |
| Scale | function | Graf3D |
| ScalePt | function | Quickdraw |
| scanBT | literal | Printing |
| scanf | function | scanf(C) |
| scanLR | literal | Printing |
| scanRL | literal | Printing |
| scanTB | literal | Printing |
| scBadParmsErr | define | Errors |
| scCommErr | define | Errors |
| scComp | define | SCSI |
| scCompareErr | define | Errors |
| scInc | define | SCSI |
| scLoop | define | SCSI |
| scMove | define | SCSI |
| scNoInc | define | SCSI |
| scNop | define | SCSI |
| scPhaseErr | define | Errors |
| ScrapStuff | type | Scrap |
| ScreenRes | function | ToolUtils |
| scrollBarProc | define | Controls |
| ScrollRect | function | Quickdraw |
| SCSICmd | function | SCSI |
| SCSIComplete | function | SCSI |
| SCSIGet | function | SCSI |
| SCSIRBlind | function | SCSI |
| SCSIRead | function | SCSI |
| SCSIReset | function | SCSI |
| SCSISelect | function | SCSI |
| SCSIStat | function | SCSI |
| SCSIWBlind | function | SCSI |
| SCSIWrite | function | SCSI |
| scStop | define | SCSI |

| Identifier | Type | Manual page |
|--------------|----------|-------------|
| secLeadingZ. | define | Packages |
| Secs2Date | function | OSUtils |
| sectNFErr | define | Errors |
| SectRect | function | Quickdraw |
| SectRgn | function | Quickdraw |
| SeedFill | function | Quickdraw |
| seekErr | define | Errors |
| select | define | Desk |
| selectMsg | define | Devices |
| SelectWindow | function | Windows |
| SelIText | function | Dialogs |
| SendBehind | function | Windows |
| SENoDB | define | Errors |
| SerClrBrk | function | Serial |
| SerGetBuf | function | Serial |
| SerHShake | function | Serial |
| SerReset | function | Serial |
| SerSetBrk | function | Serial |
| SerSetBuf | function | Serial |
| SerShk | type | Serial |
| SerStaRec | type | Serial |
| SerStatus | function | Serial |
| SetApplBase | function | Memory |
| SetApplLimit | function | Memory |
| setbuf | function | setbuf (C) |
| SetClikLoop | function | TextEdit |
| SetClip | function | Quickdraw |
| SetCRefCon | function | Controls |
| SetCTitle | function | Controls |
| SetCtlAction | function | Controls |
| SetCtlMax | function | Controls |
| SetCtlMin | function | Controls |
| SetCtlValue | function | Controls |
| SetCursor | function | Quickdraw |

| Identifier | Type | Manual page |
|----------------|----------|-------------|
| SetDAFont | function | Dialogs |
| SetDateTime | function | OSUtils |
| SetDItem | function | Dialogs |
| SetEmptyRgn | function | Quickdraw |
| setenvironment | function | SANE |
| SetEOF | function | Files |
| SetEventMask | function | OSEvents |
| setexception | function | SANE |
| SetFInfo | function | Files |
| SetFLock | function | Files |
| SetFPos | function | Files |
| SetGrowZone | function | Memory |
| sethalt | function | SANE |
| sethaltvector | function | SANE |
| SetHandleSize | function | Memory |
| SetItem | function | Menus |
| SetItemIcon | function | Menus |
| SetItemMark | function | Menus |
| SetItemStyle | function | Menus |
| SetIText | function | Dialogs |
| setjmp | function | setjmp (C) |
| SetMenuBar | function | Menus |
| SetMenuFlash | function | Menus |
| SetOrigin | function | Quickdraw |
| SetPenState | function | Quickdraw |
| SetPort | function | Quickdraw |
| SetPort3D | function | Graf3D |
| SetPortBits | function | Quickdraw |
| setprecision | function | SANE |
| SetPt | function | Quickdraw |
| SetPt2D | function | Graf3D |
| SetPt3D | function | Graf3D |
| SetPtrSize | function | Memory |
| SetRect | function | Quickdraw |

| Identifier | Type | Manual page |
|-----------------|----------|-------------|
| SetRectRgn | function | Quickdraw |
| SetResAttrs | function | Resources |
| SetResFileAttrs | function | Resources |
| SetResInfo | function | Resources |
| SetResLoad | function | Resources |
| SetResPurge | function | Resources |
| setround | function | SANE |
| SetStdProcs | function | Quickdraw |
| SetString | function | ToolUtils |
| SetTagBuffer | function | disks |
| SetTime | function | OSUtils |
| SetTrapAddress | function | OSUtils |
| setvbuf | function | setbuf(C) |
| SetVol | function | Files |
| SetWindowPic | function | Windows |
| SetWordBreak | function | TextEdit |
| SetWRefCon | function | Windows |
| SetWTitle | function | Windows |
| SetZone | function | Memory |
| SFGetFile | function | Packages |
| SFPPGetFile | function | Packages |
| SFPPPutFile | function | Packages |
| SFPutFile | function | Packages |
| SFReply | type | Packages |
| SFTypeList | type | Packages |
| shadow | define | Quickdraw |
| ShieldCursor | function | ToolUtils |
| shiftKey | define | Events |
| shortDate | literal | Packages |
| ShowControl | function | Controls |
| ShowCursor | function | Quickdraw |
| ShowDItem | function | Dialogs |
| ShowHide | function | Windows |
| ShowPen | function | Quickdraw |

| Identifier | Type | Manual page |
|----------------|----------|-------------|
| ShowWindow | function | Windows |
| SIGALLSIGS | define | signal(C) |
| SIG_DFL | define | signal(C) |
| _sig_dfl | function | signal(C) |
| SIGDIGN | define | SANE |
| sighold | function | signal(C) |
| SIG_IGN | define | signal(C) |
| SIGINT | define | signal(C) |
| SignalHandler | type | signal(C) |
| SignalMap | type | signal(C) |
| signum | function | SANE |
| sigpause | function | signal(C) |
| sigrelease | function | signal(C) |
| sigset | function | signal(C) |
| sin | function | trig(C) |
| sinh | function | sinh(C) |
| Size | type | Memory |
| SizeControl | function | Controls |
| SizeResource | function | Resources |
| SizeWindow | function | Windows |
| Skew | function | Graf3D |
| sktClosedErr | define | Errors |
| SlopeFromAngle | function | ToolUtils |
| SNAN | define | SANE |
| sony | literal | disks |
| SpaceExtra | function | Quickdraw |
| spdAdjErr | define | Errors |
| sPortA | literal | Serial |
| sPortB | literal | Serial |
| SPortSel | type | Serial |
| sPrDrvr | define | Printing |
| sprintf | function | printf(C) |
| sqrt | function | exp(C) |
| srand | function | rand(C) |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|------------|----------|-------------|--------------|----------|-------------|
| srcBic | define | Quickdraw | str2dec | function | SANE |
| srcCopy | define | Quickdraw | strcat | function | string(C) |
| srcOr | define | Quickdraw | strchr | function | string(C) |
| srcXor | define | Quickdraw | strcmp | function | string(C) |
| sscanf | function | scanf(C) | strcpy | function | string(C) |
| StackSpace | function | Memory | strcspn | function | string(C) |
| StageList | type | Dialogs | String | define | Types |
| statText | define | Dialogs | StringHandle | type | Types |
| Status | function | Devices | StringPtr | type | Types |
| statusErr | define | Errors | StringToNum | function | Packages |
| StdArc | function | Quickdraw | StringWidth | function | Quickdraw |
| StdBits | function | Quickdraw | strlen | function | string(C) |
| StdComment | function | Quickdraw | strncat | function | string(C) |
| stderr | define | stdio(C) | strncmp | function | string(C) |
| stdFile | define | Packages | strncpy | function | string(C) |
| StdGetPic | function | Quickdraw | strpbrk | function | string(C) |
| stdin | define | stdio(C) | strrchr | function | string(C) |
| StdLine | function | Quickdraw | strspn | function | string(C) |
| stdout | define | stdio(C) | strtok | function | string(C) |
| StdOval | function | Quickdraw | strtol | function | strtol(C) |
| StdPoly | function | Quickdraw | StuffHex | function | Quickdraw |
| StdPutPic | function | Quickdraw | Style | type | Types |
| StdRect | function | Quickdraw | SubPt | function | Quickdraw |
| StdRgn | function | Quickdraw | swMode | define | Sound |
| StdRRect | function | Quickdraw | swOverrunErr | define | Errors |
| StdText | function | Quickdraw | symbol | define | Fonts |
| StdTxMeas | function | Quickdraw | SysBeep | function | OSUtils |
| StillDown | function | Events | SysError | function | Errors |
| stop10 | define | Serial | SysParmType | type | OSUtils |
| stop15 | define | Serial | sysPatListID | define | ToolUtils |
| stop20 | define | Serial | SysPPtr | type | OSUtils |
| StopAlert | function | Dialogs | SystemClick | function | Desk |
| stopIcon | define | Dialogs | SystemEdit | function | Desk |
| Str255 | type | Types | SystemEvent | function | Desk |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|----------------|----------|-------------|---------------|----------|-------------|
| systemFont | define | Fonts | TENew | function | TextEdit |
| SystemMenu | function | Desk | TEPaste | function | TextEdit |
| SystemTask | function | Desk | TEPinScroll | function | TextEdit |
| SystemZone | function | Memory | TEPtr | type | TextEdit |
| tan | function | trig(C) | TERec | type | TextEdit |
| tanh | function | sinh(C) | terminate | define | Desk |
| tATPAddRsp | literal | AppleTalk | terminateMsg | define | Devices |
| tATPGetRequest | literal | AppleTalk | TEScrapHandle | function | TextEdit |
| tATPRequest | literal | AppleTalk | TEScroll | function | TextEdit |
| tATPResponse | literal | AppleTalk | TESelView | function | TextEdit |
| tATPSdRsp | literal | AppleTalk | TESetJust | function | TextEdit |
| tATPSndRequest | literal | AppleTalk | TESetScrapLen | function | TextEdit |
| tDDPRead | literal | AppleTalk | TESetSelect | function | TextEdit |
| tDDPWrite | literal | AppleTalk | TESetText | function | TextEdit |
| TEActivate | function | TextEdit | testCntl | define | Controls |
| TEAutoView | function | TextEdit | TestControl | function | Controls |
| TECalText | function | TextEdit | testexception | function | SANE |
| TEClick | function | TextEdit | testhalt | function | SANE |
| TECopy | function | TextEdit | TESysJust | define | Menus |
| TECut | function | TextEdit | TEToScrap | function | TextEdit |
| TEDeactivate | function | TextEdit | TEUpdate | function | TextEdit |
| TEDelete | function | TextEdit | TextBox | function | TextEdit |
| TEDispose | function | TextEdit | TextFace | function | Quickdraw |
| TEFromScrap | function | TextEdit | TextFont | function | Quickdraw |
| TEGetScrapLen | function | TextEdit | textMenuProc | define | Menus |
| TEGetText | function | TextEdit | TextMode | function | Quickdraw |
| TEHandle | type | TextEdit | TextSize | function | Quickdraw |
| TEIdle | function | TextEdit | TextWidth | function | Quickdraw |
| TEInit | function | TextEdit | TFeed | type | Printing |
| TEInsert | function | TextEdit | TFSID | define | SCSI |
| teJustCenter | define | TextEdit | THPrint | type | Printing |
| teJustLeft | define | TextEdit | thumbCntl | define | Controls |
| teJustRight | define | TextEdit | THz | type | Memory |
| TEKey | function | TextEdit | TickCount | function | Events |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|----------------|----------|-------------|---------------|----------|-------------|
| Times | define | Fonts | TPrXInfo | type | Printing |
| TIOPUSH | define | ioctl(C) | TrackBox | function | Windows |
| TIOCPORT | define | ioctl(C) | TrackControl | function | Controls |
| TIOSPORT | define | ioctl(C) | TrackGoAway | function | Windows |
| tkBadErr | define | Errors | Transform | function | Graf3D |
| tlAPRead | literal | AppleTalk | Translate | function | Graf3D |
| tlAPWrite | literal | AppleTalk | TrapType | type | OSUtils |
| tmErr | define | Errors | trFunc | define | Packages |
| TMTask | type | Time | true | literal | Types |
| undoErr | define | Errors | TScan | type | Printing |
| tNBPCConfirm | literal | AppleTalk | twoSideErr | define | Errors |
| tNBPLookUp | literal | AppleTalk | UNDERFLOW | define | SANE |
| tNBPreRegister | literal | AppleTalk | underline | define | Quickdraw |
| toascii | macro | conv(C) | ungetc | function | ungetc(C) |
| tolower | function | conv(C) | unimpErr | define | Errors |
| _tolower | macro | conv(C) | UnionRect | function | Quickdraw |
| TONEAREST | define | SANE | UnionRgn | function | Quickdraw |
| ToolTrap | literal | OSUtils | UniqueID | function | Resources |
| tooManyReqs | define | AppleTalk | UniqueID | function | Resources |
| tooManySkts | define | AppleTalk | unitEmptyErr | define | Errors |
| TooMem | function | Memory | unlink | function | unlink(C) |
| toonto | define | Fonts | UnloadScrap | function | Scrap |
| toupper | function | conv(C) | UnloadSeg | function | SegLoad |
| _toupper | macro | conv(C) | UnmountVol | function | Files |
| TOWARDZERO | define | SANE | UNORDERED | define | SANE |
| TPPrint | type | Printing | UnpackBits | function | ToolUtils |
| TPPrPort | type | Printing | updateEvt | define | Events |
| TPRect | type | Printing | updateMask | define | Events |
| TPrInfo | type | Printing | UpdateResFile | function | Resources |
| TPrint | type | Printing | UpdtControl | function | Controls |
| TPrJob | type | Printing | UpdtDialog | function | Dialogs |
| TPrPort | type | Printing | UprString | function | OSUtils |
| TPStatus | type | Printing | UPWARD | define | SANE |
| TPStl | type | Printing | useAsync | define | OSUtils |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|----------------|----------|-------------|---------------|----------|-------------|
| useATalk | define | OSUtils | verSpain | define | Packages |
| useFree | define | OSUtils | verSweden | define | Packages |
| useResFile | function | Resources | verTurkey | define | Packages |
| userItem | define | Dialogs | verUS | define | Packages |
| userKind | define | Windows | verYugoslavia | define | Packages |
| useWFont | define | Controls | ViewAngle | function | Graf3D |
| ValidRect | function | Windows | ViewPort | function | Graf3D |
| ValidRgn | function | Windows | VInstall | function | Retrace |
| vAxisOnly | define | Controls | vLckdErr | define | Errors |
| vAxisOnly | define | Windows | volOffLinErr | define | Errors |
| VBLTask | type | Retrace | volOnLinErr | define | Errors |
| VCS | type | Files | VolumeParam | type | Files |
| venice | define | Fonts | VRemove | function | Retrace |
| verArabia | define | Packages | vType | literal | OSUtils |
| verAustralia | define | Packages | vTypeErr | define | Errors |
| verBelgiumLux | define | Packages | WaitMouseUp | function | Events |
| verBritain | define | Packages | watchCursor | define | ToolUtils |
| verCyprus | define | Packages | Wave | type | Sound |
| verDenmark | define | Packages | wCalcRgns | define | Windows |
| verFinland | define | Packages | wDispose | define | Windows |
| verFrance | define | Packages | WDPBRec | type | Files |
| verFrCanada | define | Packages | wDraw | define | Windows |
| verFrSwiss | define | Packages | wDrawGIcon | define | Windows |
| verGermany | define | Packages | wGrow | define | Windows |
| verGreece | define | Packages | wHit | define | Windows |
| verGrSwiss | define | Packages | whiteColor | define | Quickdraw |
| verIceland | define | Packages | WidthTable | type | Fonts |
| verIsrael | define | Packages | wInContent | define | Windows |
| verItaly | define | Packages | WindowPeek | type | Windows |
| verJapan | define | Packages | WindowPtr | type | Windows |
| verMalta | define | Packages | WindowRecord | type | Windows |
| verNetherlands | define | Packages | wInDrag | define | Windows |
| verNorway | define | Packages | wInGoAway | define | Windows |
| verPortugal | define | Packages | wInGrow | define | Windows |

| Identifier | Type | Manual page | Identifier | Type | Manual page |
|---------------|----------|-------------|------------|------|-------------|
| winZoomIn | define | Windows | | | |
| winZoomOut | define | Windows | | | |
| winW | define | Windows | | | |
| winWHL | define | Windows | | | |
| wPrErr | define | Errors | | | |
| wrgVclTypErr | define | Errors | | | |
| write | function | write(C) | | | |
| WriteParam | function | OSUtils | | | |
| WriteResource | function | Resources | | | |
| writErr | define | Errors | | | |
| writTermErr | define | Errors | | | |
| wrUnderrun | define | Errors | | | |
| WStateData | type | Windows | | | |
| X2Fix | function | FixMath | | | |
| X2Frac | function | FixMath | | | |
| XfMatrix | type | Graf3D | | | |
| xOfWassSend | define | Serial | | | |
| XerRgn | function | Quickdraw | | | |
| Yaw | function | Graf3D | | | |
| yellowBit | define | Quickdraw | | | |
| yellowColor | define | Quickdraw | | | |
| ym.. | define | Packages | | | |
| ZERONUM | define | SANE | | | |
| ZeroScrap | function | Scrap | | | |
| Zone | type | Memory | | | |
| ZoomWindow | function | Windows | | | |



Appendix D



Graf3D: Three-Dimensional Graphics

Graf3D is a C library that uses QuickDraw calls to produce three-dimensional graphics. It does this by providing a fixed-point interface to QuickDraw's integer coordinates.

The Graf3D routines provide several important features:

- A camera's-eye view. This allows you to set the point of view from which the observer sees the object independently from the coordinates of the object itself. The camera is set up with the `ViewPort`, `LookAt`, and `ViewAngle` functions. You can set the focal length of the camera as if you had a choice of telephoto, wide-angle, or normal lenses.
- Three-dimensional clipping to a true pyramid. The apex of the pyramid is at the point of the camera eye, and the base of the pyramid is equivalent to the `ViewPort`. When you use the `Clip3D` function, only objects in front of the camera eye and within the pyramid are displayed on the screen.
- Two-dimensional point and line capability using fixed-point coordinates. Graf3D provides commands corresponding to the QuickDraw commands but using fixed-point coordinates instead of integers. With fixed-point coordinates you have a larger dynamic range for graphics calculations; with integer coordinates you get faster drawing time.
- Two-dimensional or three-dimensional rotation. You can rotate an object along any or all axes simultaneously, using the `Pitch`, `Yaw`, and `Roll` functions.
- Translation and scaling of objects in one or more axes simultaneously. *Translation* means movement anywhere in three-dimensional space. *Scaling* means shrinking or expanding.

How to use Graf3D

To use Graf3D, do the following:

1. Include the statements

```
#include <Types.h>
#include <QuickDraw.h>
#include <Graf3D.h>
```

in your source text.

2. Link your object file with the file {Libraries} Interface.o.
3. Set values in the Graf3D data structures and call the Graf3D routines from your program, following the information given below.

Graf3D data types

Graf3D declares and uses these data types:

```
Fixed
Point3D
Point2D
XfMatrix
Port3DPtr
```

These types are discussed below.

Point3D

```
typedef struct Point3D {
    Fixed    x, y, z;
} Point3D;
```

Point3D contains three fixed-point coordinates: *x*, *y*, and *z*. Graf3D uses *x*, *y*, and *z* for fixed-point coordinates to distinguish between the *h* and *v* integer screen coordinates used by QuickDraw.

Point2D

```
typedef struct Point2D {
    Fixed    x, y;
} Point2D;
```

Point2D is just like a Point3D but contains only x and y coordinates.

XfMatrix

```
typedef Fixed XfMatrix[4][4];
```

The XfMatrix is a 4x4 matrix of Fixed values, used to hold a transformation equation. Each transforming routine alters this matrix so that it contains the concatenated effects of all transformations applied.

Port3D, Port3DPtr

```
typedef struct Port3D {
    GrafPtr  grPort;
    Rect      viewRect;
    Fixed     xLeft, yTop, xRight, yBottom;
    Point3D   pen, penPrime, eye;
    Fixed     hSize, vSize;
    Fixed     hCenter, vCenter;
    Fixed     xCotan, yCotan;
    char      filler;
    char      ident;
    Fixed     xForm[4][4];
} Port3D, *Port3DPtr;
```

The type Port3DPtr contains all the state variables needed to map fixed-point coordinates into integer screen coordinates. These are the variables:

| Name | Description |
|---------------------------------|--|
| GrPort | Pointer to the grafPort associated with this Port3D |
| viewRect | Viewing rectangle within the grafPort; the base of the viewing pyramid |
| xLeft, yTop, xRight, yBottom | World coordinates corresponding to the viewRect |
| pen | Three-dimensional pen location |
| penPrime | Pen location transformed by the xForm matrix |
| eye | Three-dimensional viewpoint location established by ViewAngle |
| hSize, vSize | Half-width and half-height of the viewRect in screen coordinates |
| hCenter, vCenter | Center of the viewRect in screen coordinates |
| xCotan, yCotan | Viewing cotangents set up by ViewAngle, used by Clip3D |

| | |
|---------------------|--|
| <code>ident</code> | Boolean variable (stored as a char) that allows the transformation to be skipped when <code>xForm</code> is an identity matrix |
| <code>filler</code> | Char filler field to match Pascal format |
| <code>xForm</code> | 4x4 matrix that holds the net result of all transformations |

Graphics functions

Graf3D provides several functions to establish a graphics environment and create drawings within it. They are described in this section.

The InitGraf3D function

```
pascal void InitGrf3D(port)
    Port3DPtr *port;
```

The InitGraf3D function initializes the Port3D variable. Call this routine before doing Graf3D operations. You must allocate space for a variable of type Port3DPtr (whose address is passed as a parameter to this function).

The Open3DPort function

```
pascal void Open3DPort(port)
    Port3DPtr port;
```

The Open3DPort function initializes all the fields of a Port3D to their defaults, and makes that Port3D the current one. Grport is set to the currently open grafPort. These are the default values:

```
Open3DPort Function p. D-19
thePort3D = port;
port->grPort = qd.thePort;
Viewport(qd.thePort->portRect);
LookAt(qd.thePort->portRect.left, qd.thePort->portRect.top,
    qd.thePort->portRect.right, qd.thePort->portRect.bottom);
ViewAngle(X2Fix(0.0));
Identity;
MoveTo3D(X2Fix(0.0), X2Fix(0.0), X2Fix(0.0));
```

The SetPort3D function

```
pascal void SetPort3D(port)
```

```
Port3DPtr port;
```

The SetPort3D function makes port the current Port3D and calls SetPort for that Port3D's associated grafPort. SetPort3D allows an application to use more than one Port3D and switch between them.

The GetPort3D function

```
pascal void GetPort3D(port)
    Port3D *port;
```

The GetPort3D function returns a pointer to the current Port3D. This function is useful when you are using several Port3Ds and want to save and restore the current one.

The Move functions

Graf3D provides four Move functions:

```
pascal void MoveTo2D(x, y)
    Fixed    x, y;
pascal void MoveTo3D(x, y, z)
    Fixed    x, y, z;
pascal void Move2D(x, y)
    Fixed    x, y;
pascal void Move3D(x, y, z)
    Fixed    x, y, z;
```

These functions move the pen in two or three dimensions without drawing lines. The fixed-point coordinates are transformed by the xForm matrix and projected onto flat screen coordinates; then Graf3D calls QuickDraw's MoveTo function with the result.

The Line functions

Graf3D provides four Line functions:

```
pascal void LineTo2D(x, y)
    Fixed    x, y;
pascal void LineTo3D(x, y, z)
    Fixed    x, y, z;
pascal void Line2D(x, y)
    Fixed    x, y;
pascal void Line3D(x, y, z)
    Fixed    x, y, z;
```

These functions draw two- and three-dimensional lines from the current pen location. The `LineTo2D` and `LineTo3D` functions stay on the same z-plane. The fixed-point coordinates are first transformed by the `xForm` matrix, then clipped to the viewing pyramid, then projected onto the flat screen coordinates and drawn by calling QuickDraw's `LineTo` function.

The Clip3D function

```
pascal short Clip3D(src1, src2, dst1, dst2)
    Point3D  *src1, *src2;
    Point    *dst1, *dst2;
```

The `Clip3D` function clips a three-dimensional line segment to the viewing pyramid and returns the clipped line projected onto screen coordinates. `Clip3D` returns true (nonzero) if any part of the line is visible. If no part of the line is within the viewing pyramid, `Clip3D` returns false (zero).

The Set Point functions

```
pascal void SetPt3D(pt3D, x, y, z)
    Point3D  *pt3D;
    Fixed    x, y, z;
pascal void SetPt2D(pt2D, x, y)
    Point2D  *pt2D;
    Fixed    x, y;
```

The `SetPt3D` function assigns three fixed-point values to a `Point3D`. The `SetPt2D` function assigns two fixed-point values to a `Point2D`.

Setting up the camera

Functions `ViewPort`, `LookAt`, and `ViewAngle` position the image in the `grafPort`, aim the camera, and choose the lens focal length in order to map three-dimensional coordinates onto the flat screen space. These functions may be called in any order.

The ViewPort function

```
pascal void ViewPort(r)
    Rect    *r;
```

The ViewPort function specifies where to put the image in the grafPort. The ViewPort rectangle is in integer QuickDraw coordinates, and tells where to map the LookAt coordinates.

The LookAt function

```
pascal void LookAt(left, top, right, bottom)
    Fixed    left, top, right, bottom;
```

The LookAt function specifies the fixed-point x and y coordinates corresponding to the viewRect.

The ViewAngle function

```
pascal void ViewAngle(angle)
    Fixed    angle;
```

The ViewAngle function controls the amount of perspective by specifying the horizontal angle (in degrees) subtended by the viewing pyramid. Typical viewing angles are 0° (no perspective), 10° (telephoto lens), 25° (normal perspective of the human eye), and 80° (wide-angle lens).

The transformation matrix

The transformation matrix allows you to impose a coordinate transformation between the coordinates you plot and the viewing coordinates. Each of the transformation functions concatenates a cumulative transformation onto the xForm matrix. Subsequent lines drawn are first transformed by the xForm matrix, then projected onto the screen as specified by ViewPort, LookAt, and ViewAngle.

The Identity function

```
pascal void Identity();
```

The Identity function resets the transformation matrix to an identity matrix.

The Scale function

```
pascal void Scale(xFactor, yFactor, zFactor)
    Fixed    xFactor, yFactor, zFactor;
```

The Scale function modifies the transformation matrix so as to shrink or expand by xFactor, yFactor, and zFactor. For example, Scale(X2Fix(2.0), X2Fix(2.0), X2Fix(2.0)) will make everything come out twice as big when you draw.

The Translate function

```
pascal void Translate(dx, dy, dz)
    Fixed    dx, dy, dz;
```

The Translate function modifies the transformation matrix so as to displace by dx, dy, and dz.

The Pitch function

```
pascal void Pitch(xAngle)
    Fixed    xAngle;
```

The Pitch function modifies the transformation matrix so as to rotate xAngle degrees around the x-axis. A positive angle rotates clockwise when looking at the origin from positive x.

The Yaw function

```
pascal void Yaw(yAngle)
    Fixed    yAngle;
```

The Yaw function modifies the transformation matrix so as to rotate yAngle degrees around the y-axis. A positive angle rotates clockwise when looking at the origin from positive y.

The Roll function

```
pascal void Roll(zAngle)
    Fixed    zAngle;
```

The Roll function modifies the transformation matrix so as to rotate zAngle degrees around the z-axis. A positive angle rotates clockwise when looking at the origin from positive z.

The Skew function

```
pascal void Skew(zAngle)
    Fixed    zAngle;
```

The Skew function modifies the transformation matrix so as to skew `zAngle` degrees around the `z`-axis. Skew only changes the `x` coordinate; the result is much like the slant QuickDraw gives to italic characters. (`Skew(X2Fix(15.0))` makes a reasonable italic.) A positive angle rotates clockwise when looking at the origin from positive `z`.

The Transform function

```
pascal void Transform(src, dst)
    Point3D *src, *dst;
```

The Transform function applies the `xForm` matrix to `src` and returns the result as `dst`. If the transformation matrix is Identity, `dst` will be the same as `src`.

Appendix E

C Compiler Syntax

This appendix describes the syntax of MPW's C compile command.

Syntax `C [option ...] [file]`

Description Compiles the specified C source file. Compiling file *Name.c* creates object file *Name.c.o*. (By convention, C source-file names end in a ".c" suffix.) If no filenames are specified, standard input is compiled and the object file "c.o" is created.

Input If no filenames are specified, standard input is compiled. You can terminate input by typing Command-Enter.

Output If you specify the `-e` option, preprocessor output is written to standard output, and no object file is produced.

Diagnostics Errors and warnings are written to diagnostic output. If the `-p` option is specified, progress and summary information is also written to the diagnostic output.

Status The following status values are returned:

- 0 Successful error completion
- 1 Errors occurred

Options `-c` Include comments with the preprocessor output. (By default, comments are not written to the preprocessor output.)

- d *name*** Define *name* to the preprocessor with the value 1. This is the same as writing
- ```
#define name 1
```
- at the beginning of the source file. (The -d option does not override #define statements in the source file.)
- d *name=string*** Define *name* to the preprocessor with the value *string*. This is the same as writing
- ```
#define name string
```
- at the beginning of the source file.
- e** Do not compile the program. Instead, write the output of the preprocessor to standard output. This option is useful for debugging preprocessor macros.
- g** Generate stack frame pointers in A6 (that is, LINK A6, x ... UNLK A6) for all functions. Insert the procedure name into the object code that follows the procedure's RTS instruction. Use this option if you plan to debug the program with MacsBug.
- ga** Generate stack frame pointers in A6 (that is, LINK A6, x ... UNLK A6) for all functions.
- i *pathname* [, *pathname*]...**
- Search for include files in the specified directories. Multiple -i options may be specified. At most, 15 directories will be searched. The search order is as follows:
1. The include file name is used as specified. If a *full pathname* is given, then no other searching is applied.
If the file wasn't found and the pathname used to specify the file was a *partial pathname* (no colons in the name or a leading colon), then the following directories are searched:
 2. The directory containing the current input file.
 3. The directories specified in -i options, in the order listed.
 4. The directories specified in the Shell variable {CIncludes}.
- o *objname*** Pathname for the generated object file. If *objname* ends with a colon (:), it indicates a directory for the output file, whose name is then formed by the normal rules (that is, *inputFilename.o*). If *objname* does not end with a colon, the object file is written to the file *objname*.

- p Write progress information (include-file names, function names, and sizes) and summary information (number of errors and warnings, code size, global data size, compilation time, and compilation memory requirements) to diagnostic output.
- q Optimize the code for speed, even if it's necessary to make the object code larger. By default, the Compiler performs optimizations that make the code both smaller and quicker; the -q option will perform further optimizations that may make the code faster, but also larger. The -q option should be specified only for those parts of the program that are executed frequently—it's counterproductive to specify -q on code that's rarely executed.
- q2 Allow the optimizer to assume that memory locations do not change except by explicit stores; that is, the optimizer is guaranteed that (1) no memory locations are I/O registers that can be changed by external hardware, and (2) no memory locations are shared with other processes that can change them asynchronously with respect to the current process. This option must be used with extreme caution in device drivers, operating systems, and shared-memory environments, and when interrupts are present.
- s *name* Name the object code segment. (The default segment name is "Main".) Because a segment may not exceed 32K bytes, large programs require multiple segments with different names. This option is overridden if the following statement appears in the source code:


```
#define __SEG__ name
```
- u *name* Undefine the predefined preprocessor symbol *name*. This is the same as writing


```
#undef name
```

at the beginning of the source file.
- w Suppress compiler warning messages. (By default, warnings are written to diagnostic output.)
- x6 Use MOVE #0, x instructions rather than CLR x instructions for nonstack addresses. This option may be useful when writing device drivers.
- x55 Make bit fields of types int, short, and char be signed. (The default is for all fields to be unsigned.)

-z6 Always allocate 32 bits for enumerated data types, to maintain compatibility with Standard C. The default is to allocate 8, 16, or 32 bits.

Caution: This option is not compatible with the Macintosh Interface Libraries.

-z84 Enable language anachronisms. Warning messages are provided when anachronisms are encountered, and the constructs are compiled.

Example `c -p Sample.c`

Compile Sample.c, producing the object files Sample.c.o. Write progress information to the diagnostic output. (Sample.c is found in the CExamples folder.)

Limitation One MB of RAM is recommended; on a Macintosh 512K, even small C programs may not compile.



Glossary

int: A 32-bit pointer data type.

application: A program (such as the MPW Shell itself) that can be launched from the Macintosh Finder.

automatic variable: A dynamic local variable that comes into existence when a function is called and disappears when it is exited.

buffer: An area of memory allocated for reading from or writing to a file.

carriage return (\r): A control code (ASCII 13) generated by the Return key; in MPW C, equal to **newline (\n)**.

char: An 8-bit character data type whose range is -128 to 127.

command: In the Standard C Library, a parameter that tells a function which of several actions to perform; in the MPW Shell, a command name and parameters.

comp: A 64-bit SANE data type with signed integral values and one NaN.

conditional compilation: Use of preprocessor commands (**#if**, **#ifdef**, **#ifndef**, **#else**, **#endif**) to vary what is compiled depending on compile-time conditions.

C SANE Library: A set of routines that provide extended-precision mathematical functions.

C string: A sequence of characters terminated by zero byte.

denormalized number: A nonzero number that is too small for normalized representation.

desk accessory: A program that is accessed from the Apple menu and shares its runtime environment with the currently executing application.

diagnostic output: The file to which MPW tools, including the C Compiler, write error messages and progress information. Diagnostic output appears following the commands being executed in the active window by default, and can be redirected to other files, windows, and selections. In C, diagnostic output is referenced using stream **stderr**.

double: A 64-bit floating-point data type—the IEEE double type.

enum: An enumerated data type of 8, 16, or 32 bits depending on the range of the enumerated literals.

environment: In SANE, consists of rounding direction, rounding precision, exception flags, and halt settings; in MPW, consists of exported variables and signal-handling capabilities.

exception: A special condition recognized in the SANE environment: invalid operation, underflow, overflow, divide by zero, and inexact.

unsigned long: A 32-bit integer data type whose range is 0 to 4,294,967,295. Identical to unsigned int.

unsigned short: A 16-bit integer data type whose range is 0 to 65,535.

void: A data type used to declare functions that don't return a value. Void may also be used to cast expressions where values are not used.

MPW & MacApp Bug Report Form

BACKGROUND

Date _____ Version _____

AREA: Compiler: C Pascal
 Assembler
Library: C Pascal Assembly
MacApp
Shell/Editor
Tool _____
Performance _____

BUG DESCRIPTION

CONTACT INFORMATION

Name: _____ Phone/Ext. _____
Address: _____
City, State, Zip _____

Please return completed form to:
MPW Bug Report; MS 27S; Apple Computer, Inc.; 20525 Mariani Ave.; Cupertino, CA 95014

